Design and Evaluation of a Shorthand Aided Soft Keyboard

Per-Ola Kristensson

2002-08-28

LIU-KOGVET-D--02/07--SE

Master's Thesis in Cognitive Science
Department of Computer and Information Science
Linköping University
Sweden

Supervisor
Dr. Shumin Zhai
IBM Almaden Research Center

Examiner
Prof. Sture Hägglund

Abstract

A major drawback of mobile computing is the lack of an efficient text entry method. Previous attempts of handwriting/ gesture recognition or stylus soft keyboards have been shown to be inefficient, inaccurate or very hard to learn. In this work, we utilize the ATOMIK alphabetically tuned and optimized stylus soft keyboard and introduce the concept of word-level shorthand strokes formed after the corresponding keys on the keyboard layout. Since the gestures are scale and partially location independent, they require little or no visual attention from the user. An implication of Zipf's law is that the most common words make up a large percent of the text mass - in the 100 million words large British National Corpus the 100 most common individual words make up 46% of the entire corpus. Hence the largest performance gain is acquired by introducing gestures for only a limited subset of the English language and utilizing the soft keyboard for the rest. A gesture recognition engine based on elastic matching was developed and a working prototype system was used in a longitudinal pilot study involving six subjects. The study showed that the users learned about 58 shorthand gestures on average after four sessions of 40 minutes practice. Their rate of learning was rather constant across sessions, acquiring about 15 new words in each session of practice.

Acknowledgements

I would like to thank my supervisor Dr. Shumin Zhai for presenting his idea of SHARK to me and allowing me to work on it and making it a reality. I am grateful for the time he has spent and for the many invaluable comments and insights.

I also acknowledge the kind financial support I have received from my examiner Prof. Sture Hägglund of Santa Anna IT Research Institute AB.

Table of Contents

Table of Figures

Chapter 1
Introduction

1.1 Background

The concept of ubiquitous computing is a not a new one (Weiser 1991) but it has yet one major obstacle to become completely true. Although mobile phones and PDAs[1] are becoming more common, a major drawback of small mobile devices is still the lack of a truly effective text entry method with comparable speed as the QWERTY PC keyboard. A recent market analysis report[2] suggests that messaging will be the key product for future wireless data applications. The requirement for *small* yet *productive* mobile devices is a problem that as of today has not been solved.

During the years, researchers have developed handwriting recognition engines that at the current state can be said to be sufficiently effective for simple note taking. The problem of handwriting recognition is the low human performance. Natural handwriting has been found to have a peek rate of about 33 wpm (MacKenzie, Nonnecke, Riddersma, McQueen, et al 1994). Speech recognition could be the solution, but studies have shown that the text entry rate for speech is even lower (Karat, Halverson, Horn & Karat 1999).

A possible solution could be to use the established shorthand systems of Gregg and Pitman respectively, which where originally used by secretaries for taking notes during meetings. Shorthand is however very difficult to learn and users often require professional training to be truly effective. Moreover, shorthand notes are much harder to recognize than cursive script, which today's systems are hardly able to recognize with acceptable accuracy.

---

[1] Personal Digital Assistants, i.e. handheld computers.
[2] Gartner Dataquest, April 25th 2002.

As a result of the lack of a viable text entry system for mobile devices, HCI researchers have invented new systems. As we will see, most of these systems are either slow, difficult to learn or both.

1.2 Research Questions

In this thesis, we introduce the concept of word level shorthand gestures, which are constructed after a speed-wise optimized keyboard layout. The gestures may be written in a scale and partially location independent way, reducing the need for visual attention. This, and the fact that the gestures are constituted from the layout, suggests that gestures are a faster text entry method than the normal tapping on a soft keyboard, although this remains to be firmly concluded.

However several questions remain. In this thesis we are mainly approaching two of them:

1. *Is it possible to construct a shorthand aided soft keyboard system, where shorthand gestures are constructed from the relative position of the keys in the keyboard layout?*

2. *If so, is it possible to learn the shorthand gestures within a reasonable timeframe?*

1.3 Thesis Outline

1.3.1 Main chapters

Chapter 2, Text Entry Techniques for Mobile Devices is a survey of the current state of the art in mobile text entry.

Chapter 3, Shorthand Gestures on the Keyboard discusses the theoretical framework and the design phase that led to our proposed model for mobile text entry.

Chapter 4, Shape Recognition is a technical discussion of the filtering and recognition algorithms deployed in the research prototype product.

Chapter 5, User Study presents the results a longitudinal pilot study of the learning and memory performance in using our proposed model.

Chapter 6, Discussion and Future Work contains discussion, observations, conclusions and future research directions.

1.3.2 Appendices

Appendix A, Enhancing the Recognition Engine is a discussion of ways to improve the developed system.

Appendix B, SHARK System Architecture is a technical discussion of the developed system.

Appendix C, SHARK API contains technical documentation for integrating the developed system into other applications.

Appendix D, Words Used in the User Study is a list of the words used in the user study along with the correspondent way of entering the words with the model proposed in this thesis.

Appendix E, Top 100 Words in the BNC contains a list of the 100 highest ranked words in the British National Corpus along with the corresponding way of entering the words with model proposed in this thesis.

Appendix F, Speed Distribution in the User Study contains speed data collected in the user study.

Appendix G, Questionnaire Used in the User Study.

1.4 Conventions

1.4.1 How to interpret illustrations of strokes

In this thesis there are many illustrations of strokes. A stroke in this thesis can be seen as a hand drawn shape, i.e. it has a direction. The direction of a stroke is indicated by a solid dot, which marks the beginning of the stroke. The stroke in Figure 1-1 for example, shows a 'S' being drawn from the top to the bottom.

Figure 1-1 Example of a stroke.

1.4.2 Charts

Charts are drawn in UML, according to Rumbaugh, Blaha, Premerlani, Eddy, et al (1991).

Chapter 2
Text Entry Techniques for Mobile Devices

Several different mobile text entry techniques have been developed for mobile platforms. The case of mobile text input is interesting for several reasons. First, mobile devices should be *mobile*, i.e. the text entry method should not bulk the device. Second, the text entry method should be *effective* so it does not take excessive time and effort when working on the device. A device featuring fast text entry allows for more productivity, and also makes it possible to utilize the device in more working situations. Third, a text entry method should be *easy to learn and adapt,* since users typically don't want to spend more time than necessary to learn a new text entry system. This chapter presents and discusses the most common approaches and solutions for mobile text entry.

2.1 Miniaturized Physical Keyboard

2.1.1 QWERTY

The most straightforward solution is to scale down a standard physical PC keyboard. The QWERTY (Figure 2-1) keyboard layout is normally preserved, allowing users familiar with normal personal computer keyboards to instantly be productive. Example of mobile devices featuring such a keyboard is the Nokia 9200 Communicator series and the Blackberry PDA from Research In Motion, Inc. Other examples of scaled down physical keyboards is the Nokia 5510 cell phone where the LCD display has been located in the middle of the keyboard layout. The drawback of scaled down physical keyboards is a performance and productivity decrease due to the small keys (Goldstein, Book, Alsiö & Tessa 1999). This has been questioned at later time though; MacKenzie & Soukoreff (2002) presents a model for two-thumbed text entry on QWERTY keyboards. Based on the model, they claim a theoretical peek

rate for expert users at 60.74 wpm[1]. This number seems unreasonable high though, to our knowledge no user study have been able to show such a high wpm for thumb based QWERTY text entry. It should be noted that Goldstein et al (1999) did actual user testing.

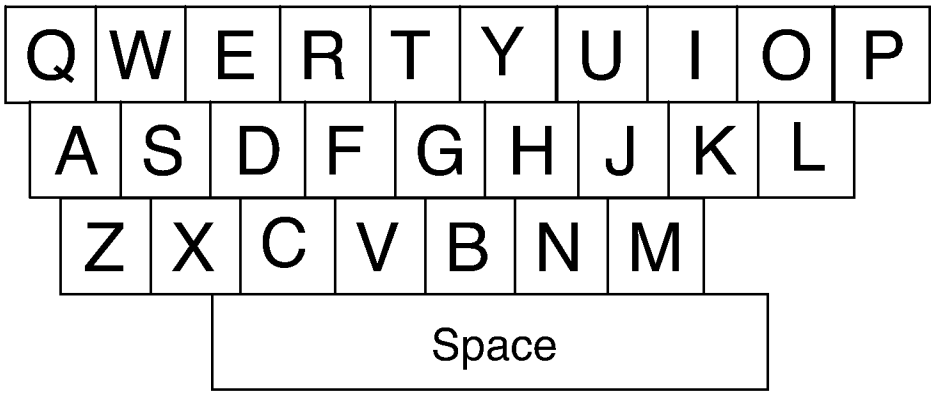


Figure 2-1 QWERTY.

2.1.2 Multitap Telephone Keypad

Another approach taken by most mobile phone producers is to utilize a telephone keypad for text entry. Since a telephone keypad only has 12 keys, several keys are ambiguous (Figure 2-2).

Figure 2-2 Telephone keypad.

This is solved by interpreting multiple key presses of the same key as different characters. Pressing the *3* button one time, outputs *d*, a second time *e*, etc. The downside with the telephone keypad is that it is slow and generally not positively received by the users (Guernsey 2000). Silfverberg, MacKenzie & Korhonen (2000) presents a model for

[1] The term *wpm* stands for words per minute. A word is defined as a sequence of five characters, including white space.

predicting text entry speed on mobile phones and suggests 25 – 27 wpm as maximum speed depending on the fingers used.

2.1.2.1 T9®

T9® is commercial product from Tegic Communications that simplifies telephone keypad text entry. Instead of tapping a button repeatedly to reach a character, T9® uses a dictionary to disambiguate the key presses, thus allowing a single tap to reach a character. Naturally, the dictionary is very limited and many button presses are ambiguous. This is resolved by a special key that alternates the different interpretations of the button presses. The candidates are shown in order of probability. In the worst case, there are so many candidates that it takes more button presses to alternate to the correct word than it takes to multitap. For this reason, the dictionary must be very limited. Silferverberg (2000) suggests a maximum speed of 41 – 46 wpm using T9®, assuming expert behavior and perfect disambiguation. It should be noted that in practice, the disambiguation is far from perfect, and MacKenzie, Kober, Smith, Jones, et al (2001) found that T9 performance decreased dramatically when the user had to alternate between the candidates.

2.1.2.2 LetterWise

An alternative to using a dictionary to disambiguate key presses is to pick the most probable key depending on the previous letters. MacKenzie et al (2001) presented LetterWise, a system that reduced the number of keystrokes by approximately 50% compared to T9.

2.2 Custom Alphabet for Stylus Text Input

Another approach is to embrace the metaphor of the PDA as an electronic calendar or notepad, and allow user to use a pen or *stylus* to enter text. Since sophisticated handwriting recognition software requires advanced hardware to reach an acceptable performance for cursive script and character recognition, custom alphabets have been developed which ease the recognition task of the device.

2.2.1 Unistrokes

Goldberg & Richardson (1993) introduces the concept of *Unistrokes*. Unistrokes are a special alphabet, designed specifically for stylus input of single letters. The design goals of Unistrokes are simple:

1. Ease the recognition process. This is accomplished by making every unistroke distinct, i.e. the prototype space should be sufficiently large, thus minimizing the confusion factor of recognition.
2. Speed. The faster the unistrokes are to write, the faster the total text input method will be.
3. Ease of learning. New user should be able to learn the new alphabet in reasonable time.

Unistrokes consists of singly connected strokes. This makes it possible to recognize individual characters on a pen down, pen up basis. Further more, Unistrokes support heads-up writing, which allows users to focus their visual attention on the application instead of the text input interface. The Unistrokes alphabet is shown in Figure 2-3.

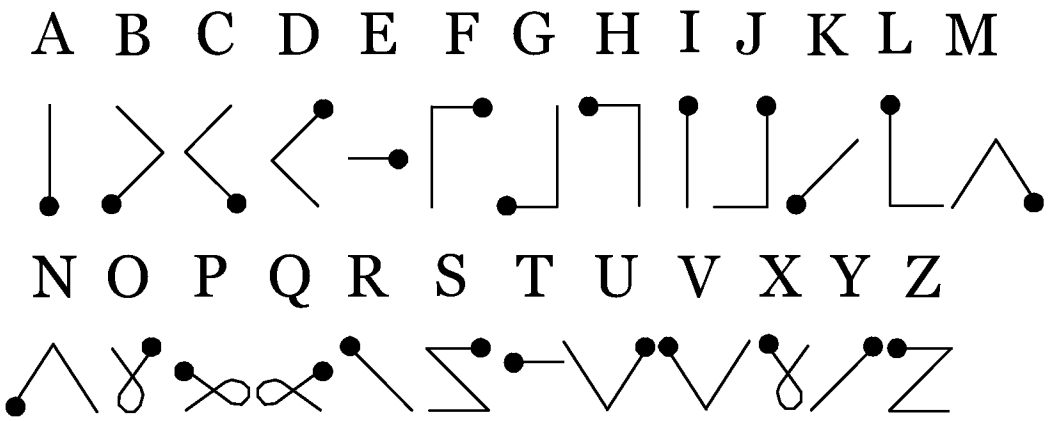


Figure 2-3 Unistrokes alphabet.

A heavy focus on the design of Unistrokes lies in speed. Goldberg et al believe that the Unistrokes alphabet lies within 10% from the maximum speed achievable in that kind of alphabet. They also did some preliminary testing that showed that users were quick to learn the new alphabet, although they initially thought it should be hard learn. Goldberg & Richardson found some recognition problems, particularly with *m* and *n* and stated that future adjustments may improve Unistrokes reliability and speed further.

2.2.2 Graffiti®

Graffiti® (Blickenstorfer 1995) is an input method similar to Unistrokes. The design goals are the same, i.e. Graffiti® is a specialized alphabet designed to be easy to recognize for the computer, easy to learn and reasonable fast to write for a user. Unlike the unistroke alphabet, the

Graffiti® alphabet is made as similar as possible to the normal Roman alphabet (Figure 2-4). This makes it easier for novice users to adapt to the system. Since the system is constrained to recognizing single characters, heads-up writing is made possible and the recognition task is significantly easier and more stable than cursive script recognition. MacKenzie & Zhang (1997) showed that users reached 97% accuracy after using Graffiti® for only 5 minutes. The drawback of Graffiti® compared to Unistrokes is the slower writing speed due to more complex strokes.

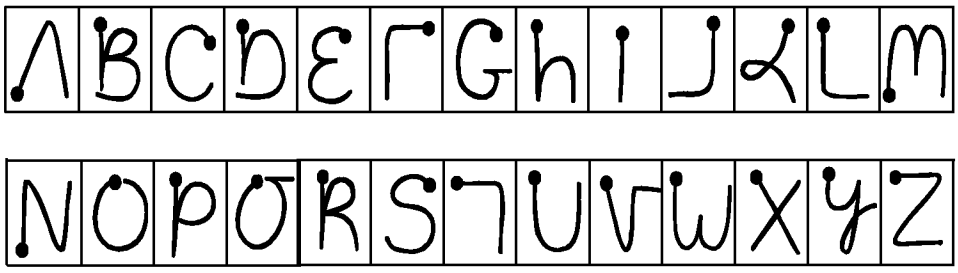


Figure 2-4 Graffiti® alphabet.

2.2.3 T-Cube

Venolia & Neiberg (1994) describes a system called T-Cube, which is a unistrokes alphabet that is accessed by the use of pie menus. The idea is that *flicks* are faster to produce. The user presses down the stylus onto one of nine target areas in a circle and depending on which area was pressed, a second pie menu divided into 8 different areas where each area contain one letter shows up. Next, the user drags the stylus to towards the desired character. As soon as the stylus has left the tablet, the corresponding character is printed to the screen (Figure 2-5). The placement of the second pie menu varies weather the user is left-handed or right-handed. This was necessary because the users hand would otherwise obstruct the second pie menu. For a right-handed user the second pie menu shows up to the upper left of the T-Cube.

There are two modes built into the system:

1. In *training mode* the corresponding pie menu shows up if the user hovers the pen slightly above a target of the T-Cube. This is feasible for novice users who often would want to search the target that brings up the correct pie menu.

2. The *expert mode* only displays the corresponding pie menu if the stylus is pressed down on the tablet. In this mode, it also possible to complete disable the pop-up of a second pie menu.

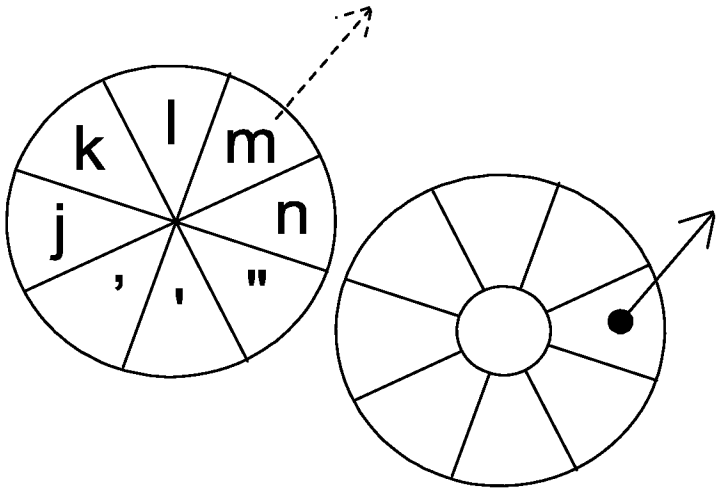


Figure 2-5 T-Cube.

Venolia & Neiberg conducted a longitudinal pilot study and compared T-Cube with other input techniques (at the time) for mobile devices. They found that T-Cube is slower than Unistrokes and soft keyboards (QWERTY). In addition they found out that the T-Cube was a little faster than handwriting (pen and paper). A drawback of T-Cube is that the system has to be learned. It took about 6 – 8 training sessions before the users were able to use the system completely in expert mode.

Another drawback of the T-Cube is the lack of heads-up writing. As Venolia & Neiberg notes, the system requires visual attention since the location of pen down determines the target and the corresponding pie menu. This, the slower speed and the long training effort are major drawbacks of the T-Cube text entry system.

2.3 Word-level Stylus Text Input

Instead of forcing the user to lift the pen between each character entered, text can be entered on the word level, similar to cursive script.

2.3.1 Cirrin

Mankoff & Abowd (1998) introduced Cirrin[2]. Cirrin consists of a soft keyboard layered out around a circle. The user draws a word by pressing down the stylus onto the middle area of the circle and dragging the stylus over the keyboard keys. When the user lifts the pen, the word is finished; hence, Cirrin operates on the word level.

Unlike Unistrokes, Graffiti and T-Cube, Cirrin operates on the word level. Naturally, the keyboard layout affects the overall speed of the system. Mankoff & Abowd created two layouts, one alphabetic and one optimized, where the most common letters were closer together and some common word combinations like *ing* were clustered. As far as we know the only evaluation of Cirrin that exists, only involves one user. Mankoff & Abowd do not mention anything about training or writing speed, except that they claim Cirrin to be about equal writing speed as other existing pen input systems mention in MacKenzie et al (1994). This claim should be taken lightly because of the lack of proper evaluation of the system.

Mankoff & Abowd also presented another version of Cirrin where the keys where laid out on two rows. To our knowledge, no evaluation of this modification exists. Figure 2-6 shows the word *finished* written using Cirrin.

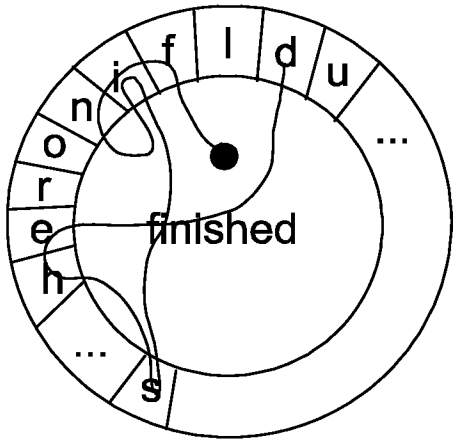


Figure 2-6 Cirrin.

[2] *Circular Input.*

2.3.2 Quikwriting

Like Cirrin, Quikwriting [sic!] (Perlin 1998) is specialized stylus text entry system that does not require pen up between characters. In fact, writing using Quikwriting can be done without lifting the pen at all.

In Quikwriting, characters are entered by moving the stylus from a central area, the *resting zone*, to one out of eight zones, and for most characters to a second zone, and finally back to the resting zone. As shown in Figure 2-7 the letter *f* is written by stroking from the resting zone into the zone containing *f*. Since *f* lies to the left of the central character in the zone, the stroke has to be completed by continuing the stroke to the zone to the left. As soon as the stroke returns to the resting zone *f* is echoed to the screen.

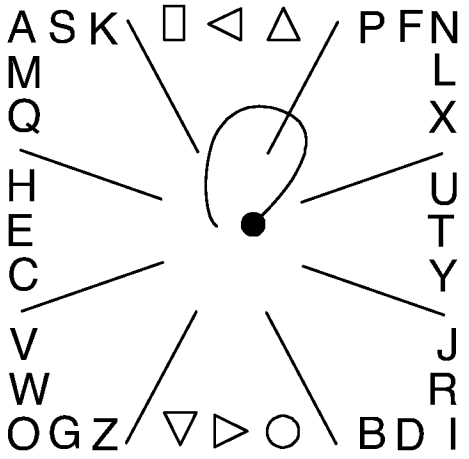


Figure 2-7 Quikwriting.

Different input modes are used for entering different character sets, i.e. *CAPITAL*, *NUMERIC*, and *PUNCTUATION*. Since the resting zone is used to identify different characters, the stylus tip never has to leave the tablet surface. One could argue that in a sense the return to resting zone requirement makes Quikwriting a specialized alphabet on the character level. To our knowledge, there is no scientific evaluation of neither the training phase or the actual speed of the system. Perlin mentions some informal subjective comments from a couple of users stating that they estimated Quikwriting as three times faster than Graffiti® and accuracy about the same. Perlin believes the speed-up compared to Graffiti® could be partially explained by certain common words becoming an *iconic gesture*, i.e. the strokes tends to get learned for common words like *the*, *and*, etc.

2.4 On-line Handwriting Recognition

On-line handwriting recognition refers to the process of identifying handwritten characters or cursive script while the user writes. Handwriting recognition has been a field of research since the early 1950's when electronic tablets first appeared (Tappert, Suen & Wakahara 1990).

2.4.1 Character Recognition

There are many methods available for character recognition (Tappert et al 1990), but most of them have required excessive processor power or suffered from low accuracy (Goldberg & Richardson 1993; Venolia & Neiberg 1994; MacKenzie et al 1994).

2.4.1.1 Jot®

Jot® is a commercial character recognition product from CIC, available for PocketPC and Palm OS® that successfully recognize a variety of strokes for a character including the strokes in natural handwriting and in Graffiti®. According to CIC, Jot is capable of recognizing up to 33 wpm. To our knowledge, no scientific evaluation of the system has been conducted. Speed wise, Jot should be up par with Graffiti® (since it recognizes the Graffiti® alphabet), and typically slower for characters more similar to the ones applied in natural handwriting. The strength of Jot® seems to be the easy adoption for novice users.

2.4.2 Cursive Script Recognition

Cursive script recognition is the "traditional" handwriting recognition problem that has been dealt with since the late 1950's (Tappert et al 1990). Several methods have been suggested and applied during the years (Tappert et al 1990; Beigi 1993). The drawback of cursive script recognition is the level of difficulty in achieving high enough accuracy; cursive script recognition is more complicated and less accurate than character recognition. Moreover, even if we assume a perfect recognizer operating at the speed of the user's natural writing, cursive script is known to have an upper limit for humans at about 33 wpm (MacKenzie et al 1994). The obvious advantage of using cursive script recognition is the immediate high level of usability, since almost all people can write. This is perhaps even truer in countries, where touch-typing is not common (Beigi 1993).

2.5 Stylus Soft Keyboards

A *soft keyboard* is a "virtual" keyboard that only exists in software. A device driver is utilized to translate the stylus coordinate data to characters. The advantage of soft keyboards is size: since the keyboard only exists on the computer screen, it does not bulk the device. In situations where no text entry is needed, the device can utilize the entire screen for optimal comfort of the user. This may be useful for instance when showing presentations, playing computer games, surfing the web and watching movies.

2.5.1 QWERTY

The first approach taken on stylus soft keyboarding is to scale down the QWERTY layout to a small screen. The QWERTY layout is very old and the design goals of QWERTY are not the same as the design goals for stylus based text entry. QWERTY was created by inventor C. L. Sholes for typewrites designed 100 years ago. Because of the way the old typewriters were constructed, the keys could jam the machine if they were pressed too closely to each other. The QWERTY layout was designed to prevent jamming, which was done by arranging the characters in such a way that the typist had to alternate between the left and right sides of the keyboard. Naturally, this is not a good design for stylus soft keyboards where the user presses the keys with a single stylus instead of using both hands. MacKenzie & Zhang (2001) studied novice users' performance on two types of soft keyboards, where one was the traditional QWERTY layout and one was a random layout. One of the user groups consisted of experienced touch typists, the other group consisted of users with moderate experience of the QWERTY keyboard. Both user groups performed about the same, with the expert touch typists gaining a slight edge on the QWERTY layout. MacKenzie & Zhang came to the conclusion that there was some skill transfer from expert touch typing on a physical QWERTY keyboard when moving on to a soft keyboard. In constrast, on a random layout the expert touch typists had basically no skill advantage at all. However, the overall improper design of the QWERTY layout for stylus text entry makes people quickly reach the low max speed.

2.5.2 FITALY

The FITALY keyboard (Figure 2-8) is a specialized soft keyboard for stylus text entry from Textware Solutions. It is available for Microsoft Windows®, Palm OS® and PocketPC. FITALY has rearranged the keys

to better fit styli text entry. Two separate space bar keys are present on the left- and right-hand sides of the keyboard respectively.

Z	V	C	H	W	K
F	I	T	A	L	Y
		N	E		
G	D	O	R	S	B
Q	J	U	M	P	X

Figure 2-8 FITALY.

2.5.3 OPTI

Using a prediction model based on Fitt's law, MacKenzie & Zhang (1999) developed a new soft keyboard (Figure 2-9). The layout was developed on a trial and error basis in a spreadsheet that calculated the maximum speed of the layout as it was modified. The keys were arranged according to the digraph frequency in the English language. Since the space bar is the most common key press, MacKenzie & Zhang decided to include four space bar keys as shown in figure 5.8. It should be noted that the advantage of multiple space bar keys is only valid if the user actually uses the optimal space bar key. This requires some planning for the user (MacKenzie & Zhang 1999).

A longitudinal training study was conducted and a peek rate of 44.3 wpm was obtained by the participants at the last session. Subject performance on OPTI layout surpassed the QWERTY layout after about 4 hours of practice. A theoretical estimation by MacKenzie & Zhang suggests a maximum speed of 58.2 wpm. This number has later been questioned (Zhai, Hunter & Smith 2000; Zhai, Sue & Accot 2002). The latest estimate (Zhai et al 2002) suggests a maximum speed of 42.8 wpm based on average Fitt's law performance. This is slightly lower than the reported peek rate of 44.3 wpm reported by MacKenzie & Zhang.

Q	F	U	M	C	K	Z
		O	T	H		
B	S	R	E	A	W	X
		I	N	D		
J	P	V	G	L	Y	F1

Figure 2-9 OPTI.

2.5.4 METROPOLIS

Zhai et al (2000) introduces another stylus soft keyboard. The layout is constructed on basis of Fitts' law movement time computations, similar to MacKenzie & Zhang (2000). However, unlike the OPTI layout Zhai et al constructs the layout using an algorithmic approach rather than trial and error. In short, Zhai et al utilized the METROPOLIS algorithm, used for searching the minimum energy state in statistical physics, by approximating the keyboard as a molecule and searched the lowest stable energy state. The energy function was said to be the mean time for typing a character, which was defined by Fitts' law (Zhai et al 2000). The result was the layout shown in figure 5.9. Notice that the vowels are connected symmetrically from the middle key.

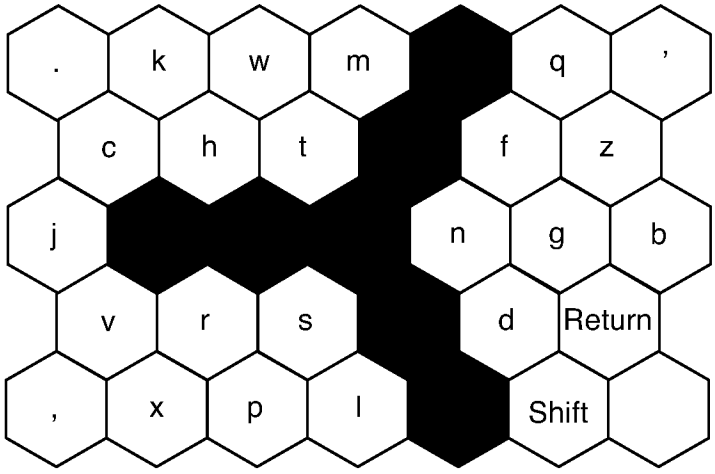


Figure 2-10 METROPOLIS.

2.5.5 ATOMIK

Training experiments conducted by Zhai et al (2002) suggested that novice users tend to prefer a more simplistic layout than the typical optimized layouts where the key placement seem chaotic at first. The idea is that an alphabetically tuned soft keyboard will limit the visual search space when the novice user scans for the correct key. This could hopefully lead to less frustration when learning the new layout. An apparent design problem is to create an alphabetically tuned layout while keeping the performance peek in speed as high as possible.

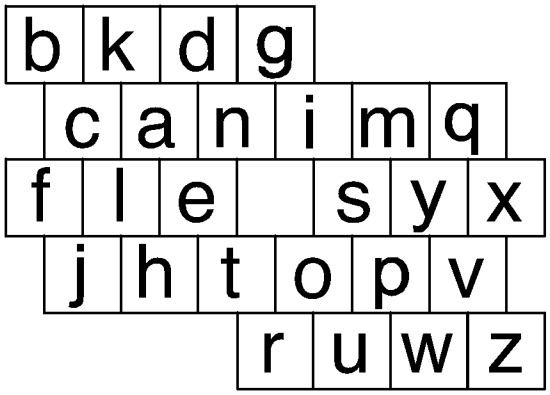


Figure 2-11 ATOMIK.

Utilizing their algorithm for generation of optimal keyboard layouts, Zhai et al produced the ATOMIK[3] layout (Figure 2-11) where the keys tend to be ordered in alphabetical layout, yet the keys are still interconnected in such a way that the performance is sustained at a very high level. Some common word combinations are grouped very closely together like 'an', 'and', 'can', 'the', 'in', 'so', 'our', etc. The estimated performance was theoretically estimated to 45.3 wpm, 1.3 wpm less than METROPOLIS but still 11.1 wpm faster than the QWERTY and 2.5 wpm faster than OPTI. An experiment conducted by Zhai & Smith (2001) showed that novice users typed 0.8 wpm faster when using an alphabetically tuned layout. When answering a questionnaire the subjects themselves also preferred the alphabetically tuned layout before an ordinary optimized soft keyboard.

[3] Alphabetically Tuned and Optimized Keyboard Layout

Chapter 3
Shorthand Gestures on the Keyboard

3.1 Brief Note on Shorthand History

Shorthand[1] was originally created to record dictations. Shorthand systems were created because normal handwriting speed was not sufficient; hence the design goal of shorthand systems has always been to speed up text entry.

The basic idea is that the text is written down with lesser strokes. The first shorthand systems invented special gestures for every word, however modern shorthand systems generally work by assigning the speech sounds special stroke segments. The most widely known shorthand systems are the ones invented by Pitman, and Gregg respectively, where Pitman's is the oldest (Oxford Encyclopedia 1998). Countries often have their own standard shorthand system; in Sweden for example the most commonly used shorthand system is the Melin system, which uses a combination of about 170 quick gestures for the most common words and a set of about 170 stroke segments for parts of speech (Nationalencyklopedin 1995).

3.2 Potential of Shorthand Text Entry

As noted by Goldberg (1993) the Gregg and Pitman shorthand systems have capacity to allow really fast text entry via styli. However, although these shorthand systems are optimized for human pen entry, they are very hard to recognize, possibly much harder than cursive script recognition. Leedhamn & Downton (1986) identifies three major obstacles for computerized shorthand entry:

1. Difficulties to construct robust recognition algorithms.
2. Ergonomic aspects of the data input device used in the survey.
3. Writers' difficulties in learning and using the system.

[1] Also known as *stenography*

Indeed, even if we assume a perfect recognition algorithm, shorthand entry would still require extensive training. Pitman's shorthand is a lot more complicated than any presented text entry system presented in the previous chapter. A particular problem regarding Pitman's shorthand is that it takes advantage of line thickness to differentiate the gestures. As shown in Figure 3-1 strokes of different weight means different things. This is not only very hard for a human to produce, it is also difficult to detect by a machine. As Leedhamn & Downton mentions, there are also some hardware issues with line thickness in this regard, since line thickness is generally regulated by electronic tablets via pen pressure. The old-fashioned pens that were in use when Pitman constructed the shorthand system made it easy to regulate line thickness by simply changing the tilt of the pen. Hence, a more sophisticated input device is probably needed to take fully advantage of an electronic version of the Pitman shorthand system.

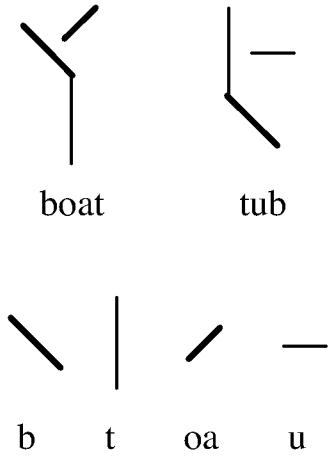


Figure 3-1 Pitman shorthand system.

However, shorthand has an extreme potential when it comes to rapid text entry, expert shorthand personnel may take notes from dictation speed at up to 120 wpm (Leedhamn, Downton, Brooks & Newell 1984).

3.3 Shorthand on a Soft Keyboard

Recognizing the fast text entry with optimized soft keyboard layouts, the question arises: "Can we do better? ". Zhai et al (2000) noted with the METROPOLIS keyboard that:

One interesting feature with the Metropolis keyboard is that the letters of some common parts of a word or an entire word, such as "the" are connected. Experienced users might be able to stroke through these letters instead of tapping on each of them. Such a strategy not only may save time but also enriches the input gestures.

This is shown in Figure 3-2.

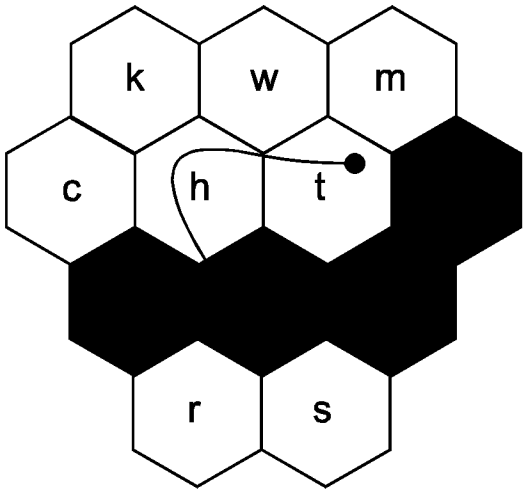


Figure 3-2 Zoom-in on a detail of METROPOLIS.

Stroking interconnected keys are one way to insert a sort of gestures to the system. However, if we limit the user to stroking exactly on the keys on the keyboard layout the task will become equal to steering through a tunnel (Accot & Zhai 1997). It will require the same amount of visual attention from the user as tapping on the keys. Moreover, only a very limited subset of the English language could be entered this way while retaining the speed performance of the soft keyboard.

The solution is to make the shorthand gestures *proportionally* bound to the keys, thus letting the user stroking them in a scale *and* location independent manner (Figure 3-3). In that case it would be viable to believe that, given the user had learned a gesture for word, he/ she would be able to stroke it faster than he/ she would be able to tap it on a soft keyboard. This speed performance is achievable because the user wouldn't need any visual attention *at all* to be able to enter the word.

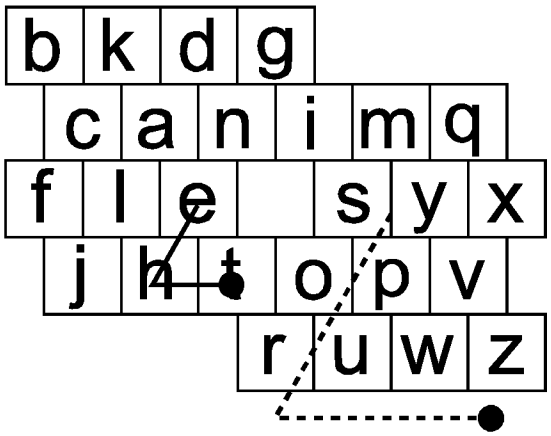


Figure 3-3 Scale and location independent strokes.

Instead of mapping the shorthand gestures to parts of speech, we are mapping them to parts of the soft keyboard. Given that the user has a pretty good idea of the layout of the soft keyboard he/ she automatically may have a clue about the geometrical shape of the gestures. In fact, the opposite might also be true: given a user learns and uses shorthand gestures formed after the soft keyboard, he/ she might develop a better sense of the location of the keys.

3.4 Scalability Issues

There are however as noted before, serious shortcomings of shorthand. The most important ones are with recognition and ease of use. A shorthand system is very difficult to recognize and perhaps even more difficult to learn to use efficiently. But we are not designing a system only consisting of shorthand gestures - the soft keyboard will still be a fallback mechanism. The question arises around how many shorthand words that should be included in the system. If we include too few words, there is no point in adopting the shorthand gestures at all since the user will be finding himself/ herself tapping all the time. If we have to include too many the users will have to devote a lot of his/ her time to train on the system. From that aspect, we have not gained much from ordinary shorthand systems.

3.4.1 Zipf's Law

G. K. Zipf stated the famous law that says that if we rank (r) words after their frequency f in a large natural text sample, we will probably find the relation:

$$f(r) \approx r^{-a}, \tag{3.1}$$

where $a \approx 1$. Zipf's law is an experimental law, it does not hold for all distributions. However for the British National Corpus (BNC), which consists of 100 million words collected from many different sources, it has been shown that the word kernel of the 10 000 most common words is a zipfian distribution with $a \approx 1$ (Figure 3-5 and Figure 3-5). In practice this means that the 100 most common words in the BNC make up close to 46% of the entire corpus. Hence, the largest performance gain is acquired by introducing shorthand gestures for only a limited subset of the English language, the 100 most common words.

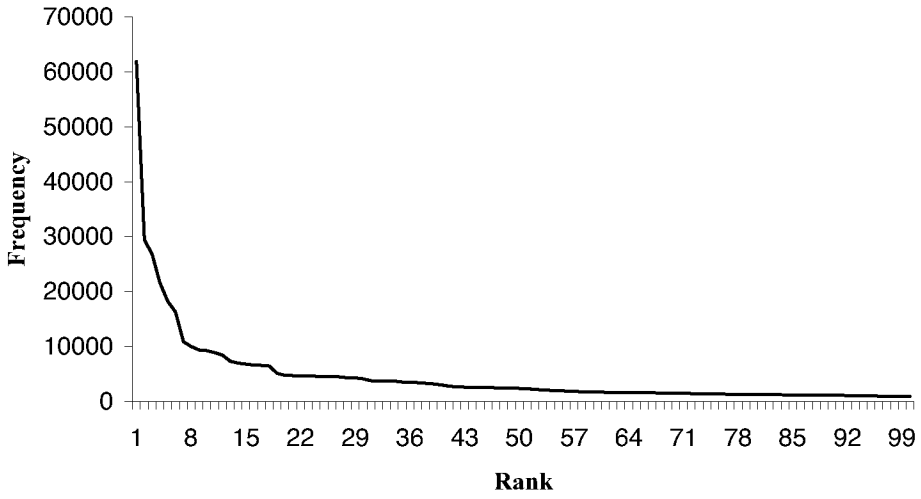


Figure 3-4 Top 100 BNC words sorted after rank.

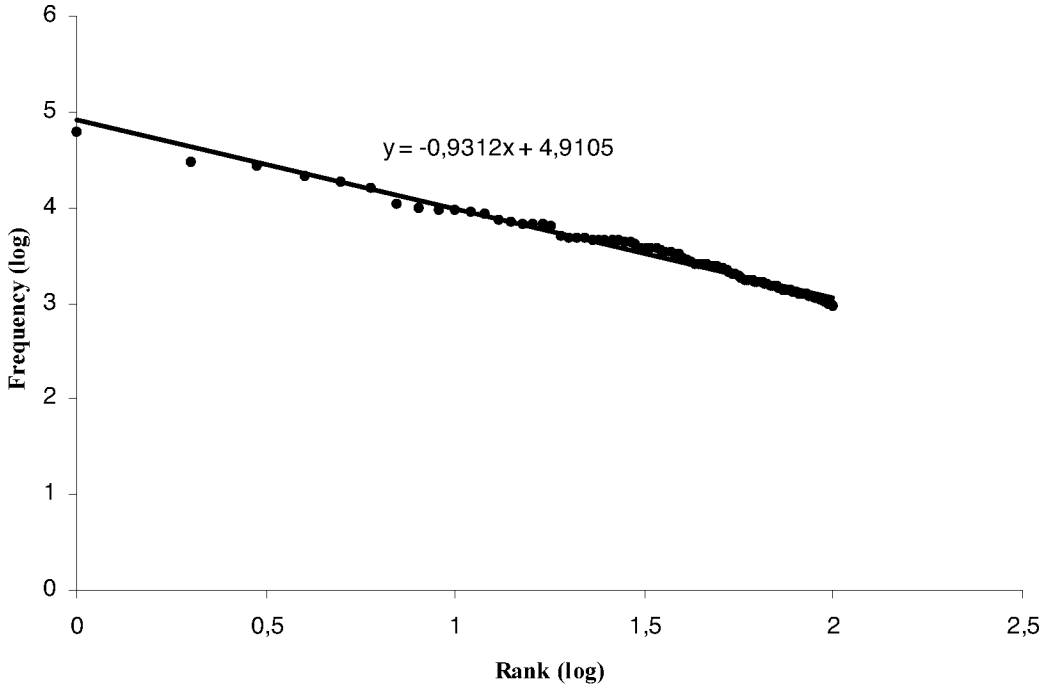


Figure 3-5 Logarithmic plot of Figure 3-4.

3.5 Resolving Ambiguity

Since the most common words in English are typically small and often share common sequences of characters (e.g. he/ she, can/ an), the gestures are bound to conflict with each other. The words 'an' and 'can' are for example completely ambiguous. There are many solutions to the problem, however a viable solution should preferably be fast and requiring as little visual attention as possible.

3.5.1 Pie Menu

One alternative would be to utilize a pie menu disambiguation system similar to the one employed by Venolia & Neiberg (1994). By identifying the ambiguous gestures (or words) and presenting them in a determined position in a pie menu a well trained user wouldn't need to glance at the keyboard when stroking an ambiguous word. He/ she would simply stroke the word knowing that the pie menu would show up immediately and then stroking a second time in the determined direction of the desired alternative (Figure 3-6).

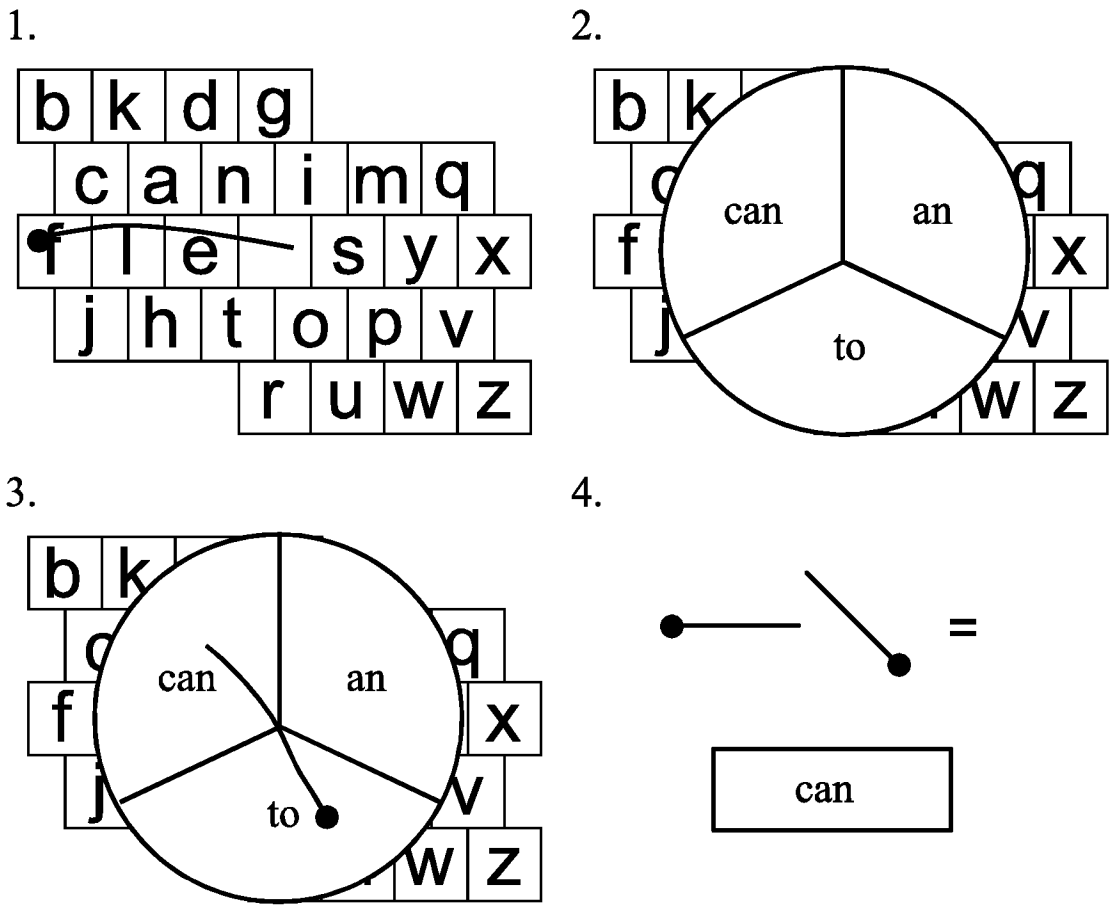


Figure 3-6 Pie menu disambiguation.

The downside of the solution is that the most ambiguous words on the ATOMIK keyboard are the shortest ones, and incidentally, the most common ones. Figure 3-7 shows the ATOMIK keyboard and highlights some very common words. In the picture *can*, *an* and *to* are completely identical if we ignore scale and location. The same goes for *do*, *is*, *my* and *no*. In addition *be* is not completely identical but very similar. All of these are very common words in English (see Appendix E for a list of the most common 100 words in the British National Corpus). Hence, it would be of disadvantage to force a second stroke, thus extending the completion time for the total gesture, for the words that are used absolutely the most.

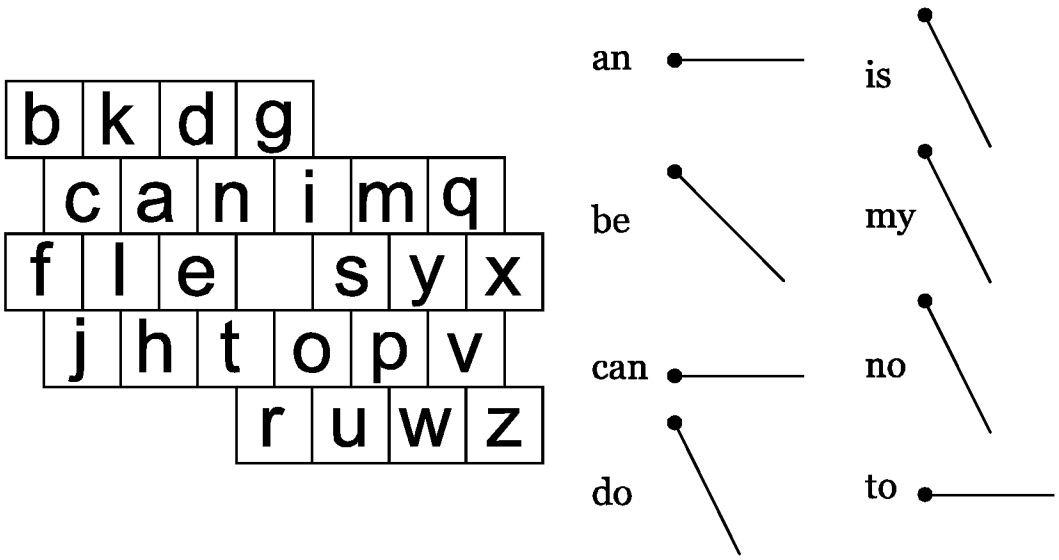


Figure 3-7 Common words which have very similar shapes.

3.5.2 Partial Location Dependency

Another solution is to calculate the Euclidean distance between the bounding box of the gesture and the ambiguous words. We then chose the word that has the least distance (Figure 3-8). A consequence is that the user only has to produce an ordinary fast gesture for a word. If the gesture is ambiguous the user has to utilize some visual attention to produce the gesture closer to the desired word. However, very little visual attention needs to be used. This is evident after studying the ATOMIK soft keyboard layout. First, very few words have a large ambiguity to begin with, secondly the user still have to use massive visual attention when tapping with the stylus on the soft keyboard, which the user will do approximately half of the time with the system anyway.

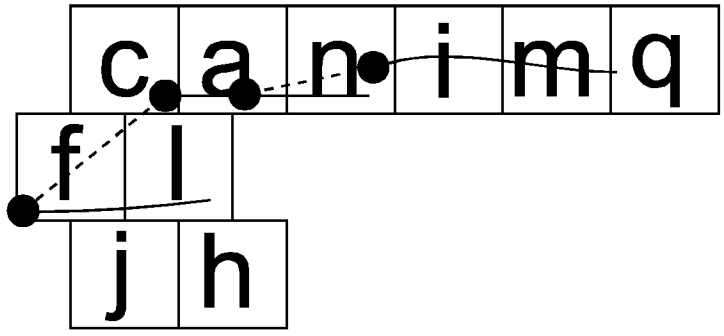


Figure 3-8 Disambiguation by location.

3.6 The SHARK System

We decided to construct a system where the shorthand gestures were formed from the key positions in the soft keyboard. We took the 100 most common words from the BNC and converted them to appropriate gestures for the system. These gestures were made scale independent. In order to disambiguate between similar gestures, we made the gestures partially location dependent. If the system found two similar words, the system should choose the word whose shape was closest to where the gesture was drawn. We call the text entry method SHARK – *Shorthand Aided Rapid Keyboarding*.

3.7 Advantages of SHARK

3.7.1 Scale Independent Word Level Gestures

Shark does share some similarity with other text entry systems for mobile devices. The most striking similarity is perhaps with Cirrin, since they both operate on the word level and create shorthand gestures out of a keyboard layout. However, since the authors of Cirrin didn't use an optimized keyboard layout the shorthand strokes of Cirrin are much longer and slower. But SHARK does not only contain faster and simpler shapes. Since we allow the user to stroke the words in *any* scale we do not constrain the user to reproduce an exact pattern on the keyboard, we only require a gesture that have some degree of spatial relation with the individual characters position on the keyboard. This allows the gestures to be learned and reproduced without knowing the exact positions of the keys. Quikwriting also allows word level gestures, but since Quikwriting forces the user to return to a large resting zone between the strokes, the gestures are not scale independent. Further more, Quikwriting gestures are also typically longer and more complicated than SHARK gestures.

3.7.2 Soft Keyboard and Word Level Gestures

SHARK does not force the user to learn a complete shorthand system, which can be tedious and time consuming. Instead the system only allows shorthand gestures for common words and provides a fallback soft keyboard mechanism for the rest. While soft keyboards built-in with additional gestures have been done at least once before[2] they have only used gestures for simple commands like space, tab and backspace. SHARK allows word level gestures, very similar to shorthand gestures.

[2] Microsoft Windows® PocketPC soft keyboard

3.7.3 Scaleable Customizable Interface

SHARK supports both the novice and the expert. The expert strokes the shorthand gestures about half time, and performs fast stylus touch tapping for the rest. The novice has to play hunt and peck in the beginning since he/ she has do not know where the keys are. But how should a novice know which words that have shorthand gestures and which words that have to be tapped?
The solution is to introduce the novice one step at a time to the system. In the beginning the novice has to tap since he/ she has no idea which words are gestures. When the novice taps a word that exists as a gesture, the system can indicate to the user that this word has an alternative shorthand gesture by animating the trace between the keys on the keyboard. Hence, time consuming training can be avoided to some extent, although a specialized training program would probably speed up the learning rate. See chapter 5, for more information regarding training.

There is no word limit to the SHARK system. Since the recognition system allows arbitrary shapes to be inserted, SHARK allows the user to insert their own words. Theoretically the user could have thousands of different words inserted, effectively creating a new input language. It could also be possible to automatically create new gestures by stroking exactly on the keys the first time. However, some special maneuver is probably needed to disambiguate such strokes.

Since the gestures are created dynamically from the keyboard layout, SHARK is independent of languages and can be localized to any language that can be entered with an ordinary soft keyboard. In fact, SHARK can run on any layout, including the traditional QWERTY layout. However, in order to achieve high stylus touch typing speed, an optimized soft keyboard is preferable.

Chapter 4
Shape Recognition

In order to properly identify the shorthand strokes on the keyboard a recognition system needed to be developed. This chapter will discuss shape recognition[1] theory and describe the actual recognition engine deployed.

4.1 Overview of Shape Recognition Theory

Ever since the electronic tablets first occurrences in the late 1950's the task of shape recognition has gained interest among researchers. An overwhelming research focus lies on handwriting recognition, particularly for large-alphabet languages like Chinese where keyboards are cumbersome (Tappert, Suen & Wakahara 1990).

4.1.1 Definitions

A *stroke*, in the handwriting recognition context, is defined as the writing from pen down to pen up. In this thesis, the term *shape* is considered equal to a stroke. That is, a shape is seen as a vector of coordinates enclosed within a pen down to pen up movement from the user. When discussing matching between a shape and a prototype, the term *curve* is used. A curve is in this context identical to a stroke.

A *prototype* is a template, i.e. the *unknown* is matched against a series of prototypes and the best *candidate* is typically returned. Some systems return an ordered list of the best candidates to be processed by higher

[1] In the context of the SHARK system, what we really mean by shape recognition is gesture recognition. However, in handwriting recognition literature gesture recognition is often given the interpretation as an interaction technique for giving commands to an application, i.e. marking words, etc.

order language engineering systems, like syntax analysis systems and probability models.

Prototype space[2] is a central term in shape recognition. Prototype space can mean different things in different approaches towards shape recognition. The common interpretation is to see a certain prototype as a *feature vector* located in an n-dimensional space and prototype space refers to the amount of difference between the prototypes feature vectors. Another interpretation is to see different prototypes as 2-dimensional geometrical forms, and refer to prototype space as an amount of shape similarity between the prototypes.

4.1.2 On-line and Off-line Recognition

Tappert et al (1990) distinguishes between two fundamental recognition paradigms. *On-line* recognition is the task of recognizing handwriting in real-time as the user writes. This is commonly done by utilizing an electronic tablet that samples the pen movement. Since the pen movement is sampled in order, on-line recognition systems have information on both the point data and the order of the points. *Off-line* recognition, on the other hand, can be performed at any time. Off-line recognition does not utilize any temporal information; therefore, off-line recognition can be used to recognize characters from other media like newspapers and printed documents. Off-line recognition is typically carried out by extracting text from a scanned image. This process is often referred to as OCR (Optical Character Recognition) (Beigi 1993).

4.1.3 Preprocessing

Due to the noisy nature of human pen input, most recognition systems utilize some sort of preprocessing on the point data. Preprocessing is mainly used to reduce the computational load and remove sample errors that could confuse the recognition algorithm (Tappert et al 1990).

Common preprocessing algorithms includes *filtering* which is used to reduce and smooth the point data, and *wild point correction* which is used to eliminate random point data which the user didn't intend to write. Wild point correction is generally caused by hardware (Tapper et al 1990). Although modern tablets connected to a personal computer have improved to such an extent that they have almost completely eliminated

[2] Sometimes referred to as sloppiness space.

the need for wild point correction, the technique has shown to be useful when moving applications to handheld computers (Aksela 2000).

Dehooking is the process of removing "hooks" in the beginning and end of a stroke. Hooks are usually caused by imprecise hardware. Since a hook in most cases doesn't contain any data that will improve recognition performance, hooks are often safely removed (Guerfali & Plamondon 1993).

Since pen input is usually carried out in an unconstrained fashion, appropriate methods for *size and location normalization* of the strokes are needed. The standard methods include scaling the shape to a uniform size and moving the shape to a predetermined location (Tappert et al 1990).

Additional preprocessing is often necessary for off-line data. Common procedures include *thinning* and *join-operations* for interconnecting segments (Dougherty 1999).

4.1.4 Recognition

Several different recognition algorithms have been developed. Most of them originate from the handwriting recognition field.

The most common techniques involved in character recognition are template matching models, syntactical models, statistical models and neural network models (Beigi 1993). Statistical models and neural network models will not be discussed further in this thesis, see Tappert et al (1990) and Beigi (1993) for reading directives regarding these models.

A template matching system has a database of templates. These templates may consist of a complete shape or parts of a handwriting segment. Different methods are used to identify the templates (Beigi 1993). One popular matching algorithm is elastic matching (Tappert 1982) which computes the minimum distance measure between two sets of points, which may or may not have the same number of points.

Another technique divides a stroke into smaller segments or features, which are matched against prototypes. These smaller segments or features can later be matched against each other using elastic matching, hidden markov models (HMM) (Tappert et al 1990) or artificial neural networks (Looney 1999).

4.1.5 Pruning

Pruning is the process of quickly filtering out prototypes, which are very unlikely to match the unknown. Due to its nature, a pruning algorithm should be conservative in the filtering process, and fast. A system may benefit a significant performance boost if the pruning algorithm is carefully engineered (Tappert et al 1990; Beigi 1993).

4.2 Recognition of Arbitrary Shapes

This section discusses the recognition of shapes in any format. The special case of recognition of simple geometrical shapes will be the subject of the next section. The only algorithms and methods discussed in this section are the ones that are implemented in the SHARK system.

4.2.1 Preprocessing

Preprocessing is the process of filtering and smoothing the input signal. Since the recognition engine receives a raw signal from a hardware device such as a mouse or an electronic tablet, the data signal contains noise which, in order to improve recognition performance, it is beneficial to filter out. Preprocessing also involves the process of normalizing the scale and location of the shape.

4.2.1.1 Filtering

Ordered data points in the form of $(x_1,y_1),(x_2,y_2),...,(x_{n-1},y_{n-1}),(x_n,y_n)$ are received from the hardware. If we let the vertex be data point p, we can compute an angle θ

$$\theta = \tan^{-1}\left(\frac{y_p - y_{p-1}}{x_p - x_{p-1}}\right),\ p>1. \tag{4.1}$$

Moreover, we can define the angle difference for two consecutive points as $\Delta\theta$

$$\Delta\theta = \theta_p - \theta_{p-1},\ p>1 \tag{4.2}$$

A sketch of the input signal is shown in Figure 4-1.

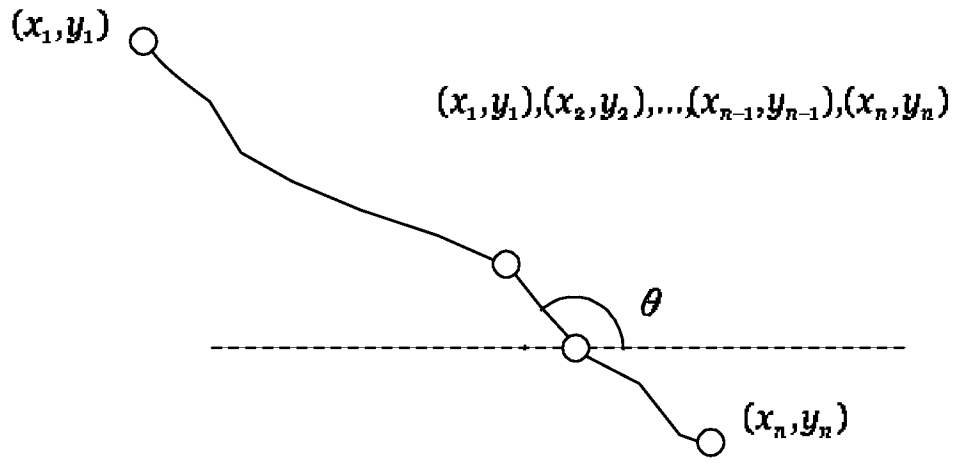


Figure 4-1 Data signal.

In order to reduce the data load for the recognition algorithm duplicate data points are removed. The algorithm also filters out data points whose Euclidean distances are below a certain threshold (Arakawa 1983). Sometimes it is advantageous to keep data points whose Euclidean distances between them are below the threshold. This could be the case if $\Delta\theta$ increase rapidly, for example, if the user did a fast twitch with the pen to indicate a corner (Tappert et al 1990). The value of the angle threshold depends on the how sensitive the filtering function should be to rapid $\Delta\theta$ changes. In the implementation an angle threshold of $\frac{\pi}{4}$ radians were used, based on experimental findings. An example of a filtering algorithm is shown in Figure 4-2.

```
function FILTER(signal, min-distance, max-angle)
    returns resampled-signal

  n←LENGTH(signal)
  Create an array resampled-signal
  resampled-signal[0]←signal[0]
  for each data point i from 1 to n do
    if EUCLIDEAN-DISTANCE(signal[i], signal[i-1]) > min-distance then
      resampled-signal[i]←signal[i]
    elseif Δθ(signal[i], signal[i-1]) > max-angle then
      resampled-signal[i]←signal[i]
  end
  return(resampled-signal)
```

Figure 4-2 Filtering algorithm.

Figure 4-3 shows the effect of the algorithm, before and after filtering. A white circle indicates the beginning of the stroke. Points that are too close to the previous points are removed. Notice that points at sharp corners are preserved despite their low distance.

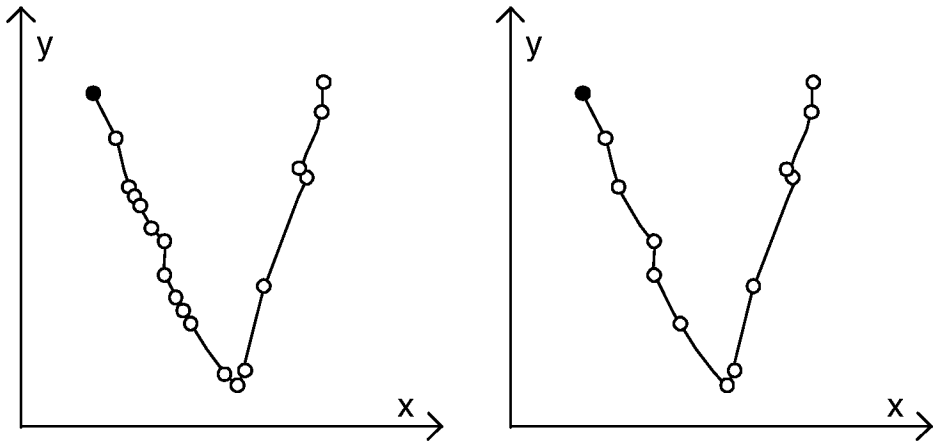


Figure 4-3 The effect of the filtering algorithm.

4.2.1.2 Normalization

Since the recognition engine is used in a context where there is no constrain on how large or where the user writes, the unknown needs to be scaled to a uniform size and centered on a common point, typically the upper-left corner or the center of the shape.

Scaling is done by computing the minimal bounding box of the curve. An operation MIN-MAX-SCALE scales the largest side of the bounding box to a predetermined size, while retaining the aspect ratio.

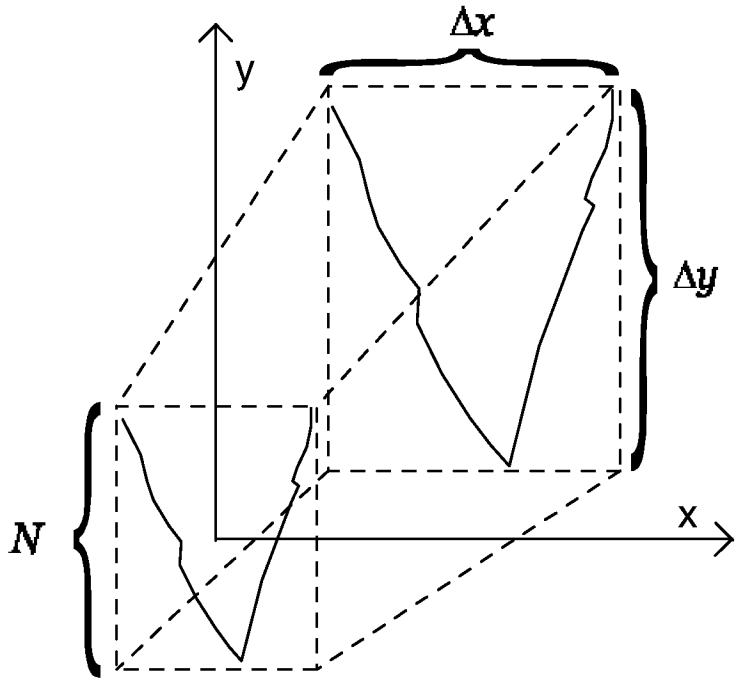


Figure 4-4 Normalization of a curve in location and scale.

Normalization of location is achieved by computing the geometrical center of the bounding box. The bounding box center is then moved to the origin of the coordinate system (Figure 4-4). Pseudo code for the implemented normalization algorithm is shown in Figure 4-5.

```
function NORMALIZE(curve) returns curve
  curve←MIN-MAX-SCALE(curve)
  curve←NORMALIZE-LOCATION(curve)
  return(curve)

procedure MIN-MAX-SCALE(curve, length) returns curve
  bounding-box←BOUNDING-BOX(curve)
  if Δx(bounding-box) > Δy(bounding-box)
    scale←length/Δx
  else
    scale←length/Δy
  return(SCALE(curve, scale))

procedure SCALE(curve, scale)
  for each point in curve do
    point←point*scale
  end

procedure NORMALIZE-LOCATION(curve) returns curve
  bounding-box-center←GET-BOUNDING-BOX-CENTER(curve)
  x←x coordinate for the first point of curve
  y←y coordinate for the first point of curve
  x-offset←x – x(bounding-box-center)
  y-offset←y – y(bounding-box-center)
  return(SET-OFFSET(curve, x-offset, y-offset))

procedure SET-OFFSET(curve, x-offset, y-offset)
  offset-length(x)←x coordinate for the first point of curve – x-offset
  offset-length(y)←y coordinate for the first point of curve – y-offset
  for each point in curve do
    point←point(x, y) + offset-length(x, y)
```

Figure 4-5 Normalization algorithm.

4.2.2 Matching

4.2.2.1 Elastic Matching of two Curves

We assume we have an unknown shape u and a set of prototypes K. We want to find the candidate in K that bests matches u, hence we pick a prototype k from K and compute a total distance D between u and k. The distance will be our similarity measure between the curves. If we put the data points relating u and k in a matrix, the total distance can be seen as a *path* in the matrix. Each cell in the matrix corresponds to a relationship between one data point in u and one data point in k. If u

and k contain n and m data points, $n \neq m$, there are many possible paths but we are only interested in the *optimal* path between u and k.

This problem can be described in mathematical terms as a minimization problem. If we express the total distance between two curves as $D(n,m;k)$ for an unknown of n points against a prototype k of m points, and express the distance between individual data points i and j as $d(i,j;k)$ (often abbreviated as $d(i,j)$) our goal will be to minimize $D(n,m;k)$ where D is:

$$D(i,j;k) = d(i,j;k) + \begin{cases} \min\begin{cases} D(i-1,j;k) \\ D(i-1,j-1;k) \\ D(i-1,j-2;k) \end{cases} j > 2 \\ \min\begin{cases} D(i-1,j;k) \\ D(i-1,j-1;k) \end{cases} j = 2 \\ \min\{D(i-1,j;k)\} j = 1 \end{cases}. \qquad (4.3)$$

Note that the definition given by Equation 4.3 ensures that all of the data points of the unknown are matched against a data point in the prototype (Scattolin 1995).

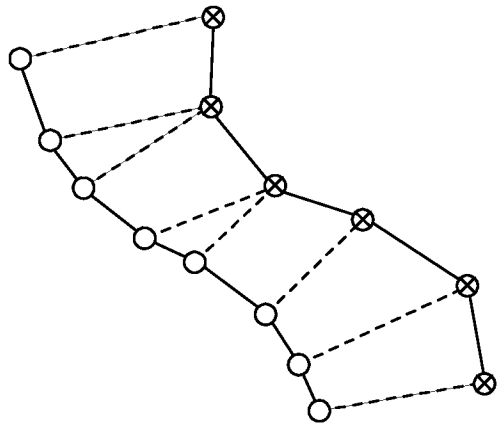


Figure 4-6 Elastic matching of two curves.

This can be solved efficiently by the use of *dynamic programming*. A matrix $[n,m]$ is constructed, where n and m is the number of data points in the curves. The optimal path is found by traversing the matrix where each column corresponds to a data point in the unknown shape and the prototype. We define a condition that states that the first and last data

points of the curves must be matched against each other respectively. In addition a data point in one curve may match several data points in the other. Each cell $[i,j]$ contains the minimum distance between the first *i*th and *j*th points. That is, $[i,j]$ contains the minimum cost from the traversed neighborhood cells plus the value of a cost function $d(i,j)$; i.e. if we are starting from the cell $[0,0]$ (Jurafsky & Martin 2000):

$$dist[i,j] = d(i,j) + \min \begin{cases} dist[i-1,j] \\ dist[i-1,j-1]. \\ dist[i,j-1] \end{cases} \tag{4.4}$$

The total distance is found in $[n,m]$. This process is also referred to as *Dynamic Time Warping* (DTW) (Tappert 1982). The generic dynamic time warping algorithm is shown in Figure 4-7.

```
function MIN-DISTANCE(unknown, prototype) returns min-distance

  n←LENGTH(unknown)
  m←LENGTH(prototype)
  Create a distance matrix distance[n+1,m+1]
  distance[0,0]←0
  for each column i from 0 to n do
    for each row j from 0 to m do
      distance[i,j]←COST[unknown_i,prototype_j] + MIN(distance[i-1,j],
                                                      distance[i-1,j-1],
                                                      distance[i,j-1])
  return(distance[n,m])
```

Figure 4-7 Dynamic time warping algorithm.

4.2.3 Distance Metrics

4.2.3.1 Point-to-point Distance

Point-to-point distance is the most simple distance metric. The cost function d can be defined as the squared Euclidean distance between data point i from the prototype and data point j from the unknown:

$$d(i,j) = (x_i - x_j)^2 - (y_i - y_j)^2 \tag{4.5}$$

An alternative is to use the Euclidean distance:

$$d(i,j) = \sqrt{(x_i - x_j)^2 - (y_i - y_j)^2}\,. \tag{4.6}$$

This puts lesser emphasis on points further away from the prototype. This metric has been shown by experimentation to receive slightly better recognition accuracy (Scattolin 1995). However, the accuracy comes with a cost of computation since calculating the Euclidean distance requires a square root operation, which is costly for computers without a FPU[3]. Today most modern desktop computers are equipped with a FPU, but on the other hand ARM, the leading processor architecture for today's (2002) handheld computers, currently comes equipped without an FPU as the default configuration (Cormie 2002), which could prove to be a serious obstacle when porting a system from research equipment to an actual product. Previous work of porting elastic matching recognition systems from research equipment to product platforms have suffered from either major slowdowns or rather extreme accuracy declines because of the target systems' lack of native floating point support (Aksela 2000).

A weakness of the distance metrics presented in Equation 4.5 and Equation 4.6 is that they do not take into account the curvature of the stroke. Hence, it can be advantageous to take the angle difference between two data points into account:

$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} + c\left|\phi_i - \phi_j\right|, \tag{4.7}$$

where c is an empirically determined constant whose value depends on the implementation, and ϕ is the absolute direction of a vector defined by two consecutive data points in the stroke (Scattolin 1995).

Several other distance metrics exists. In theory, almost any distance metric can be used. For example, Tappert (1982), who pioneered elastic matching in the context of cursive script recognition, used the slope angles and the heights of the data points as the distance metric (the users had to write on a special lined paper).

4.2.3.2 Normalized Point-to-point Distance

Since point-to-point distance is computed as the sum of the cost of all matching points in the optimal time warping path, the algorithm is

[3] Floating Point Unit.

sensitive to the number of data points in the unknown. In practice, this means that two identical strokes drawn at different speeds will have greater dissimilarity distance than two identical strokes drawn at the approximately same speed. This may be good if the system should adapt to different users that have characteristic writing styles, i.e. users tend to have individual pen velocity patterns for different characters or word segments. If an equal distance metric for identical geometric strokes is preferable, a normalized point-to-point distance, D_{NPP}, can be achieved by dividing the total cost, D_{PP}, with the total number of points matched, H, (Vuori 1999):

$$D_{NPP} = \frac{D_{PP}}{H}. \tag{4.8}$$

4.3 Recognition of Simple Shapes

In the case of the SHARK system, the prototypes are defined by vectors connecting the letters on the ATOMIK keyboard layout. Since they typically only consists of two – four data points, a direct elastic matching between the unknown and the prototypes would be uncertain and probably generate distance measures that inaccurately reflected the true relationship between the unknown and the prototypes. Moreover, later in this section we will show that the simpler shapes can be easier and more efficiently recognized using a naive matching model.

4.3.1 Non-destructive Interpolation

The unknown shapes are typically more point intensive than the prototypes; therefore, a viable solution is to interpolate approximately the same points onto the prototype as in the unknown. This can be achieved in a non-destructive way by preserving the original points of the prototype and interpolating points between these. The preferred interpolated distance d_i between the data points is

$$d_i = \frac{L_k}{n_u}, \tag{4.9}$$

where L_k is the length of prototype k and n_u is the total number of points in the unknown. Using this method, the new prototype will have $n_k + n_u$ data points, where n_k is the number of data points in the original prototype. The geometrical properties of the shape of prototype k will

remain the same. However, since the data points are interpolated at an even interval, the prototype data points may not accurately reflect the dynamic nature of the data points in the unknown. For instance, more points are generated where the pen moves slowly, typically in areas of high curvature (Tappert 1982).

A way to compute a better distance measure is to insert every interpolated data point into the prototype at the relative distance of the corresponding data point in the unknown. Using this method, it doesn't matter if the pen movement velocity changes during the stroke since the changes in velocity will be accurately reflected on the prototypes. In way the signal acts like a filter on the prototype interpolation process (Figure 4-8).

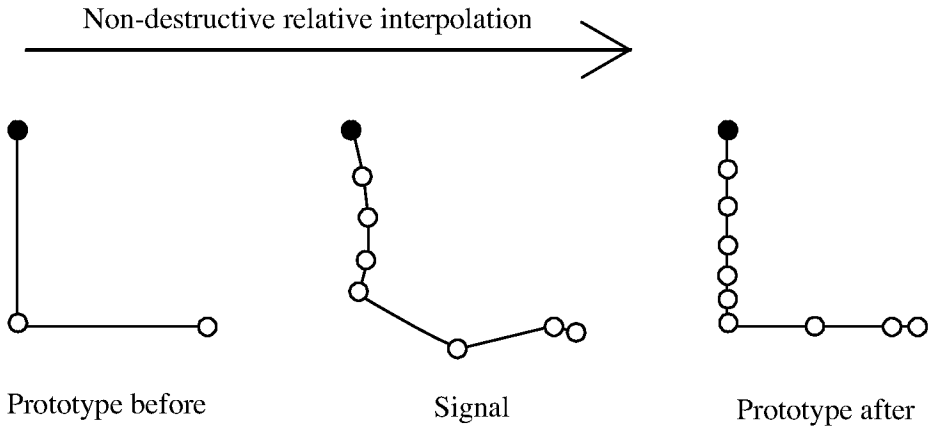


Figure 4-8 Non-destructive relative interpolation.

Relative interpolation can be achieved effectively by the use of a pre-computed look-up table containing for every data point p, $p>1$, with coordinates x and y in the unknown:

1. The Euclidean distance between the current data point and the previous:

$$d_{(p,p-1)} = \sqrt{(x_{p-1} - x_p)^2 + (y_{p-1} - y_p)^2}\ ,\ p>1. \qquad (4.10)$$

2. The total Euclidean distance so far,

$$d_{T(p)} = \sum_{i=2}^{p} \sqrt{(x_{i-1} - x_i)^2 + (y_{i-1} - y_i)^2}\ ,\ p>1. \qquad (4.11)$$

The process is shown in Figure 4-9. Now, the relative points of the unknown can be inserted into the prototype using the look-up table (Figure 4-9 and Figure 4-10).

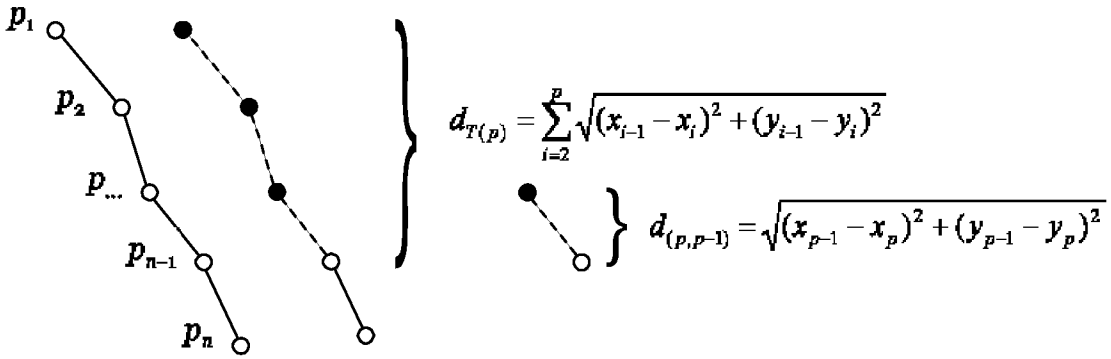


Figure 4-9 Computing the look-up table.

```
function INTERPOLATE-RELATIVELY(prototype, look-up-table)
    returns mapped-prototype

  if LENGTH(prototype) > LENGTH(look-up-table)
    return prototype
  n←LENGTH(prototype)
  D_p←EUCLIDEAN-DISTANCE(prototype)
  index←0
  create a curve mapped-prototype
  for each vector v from 0 to n do
    θ←ABSOLUTE-DIRECTION(v)
    d_v←EUCLIDEAN-DISTANCE(v)
    r← D_p / d_v
    d_t←LOCAL-DISTANCE(look-up-table[index]) / r
    D_t←(EUCLIDEAN-DISTANCE(LENGTH(look-up-table, index))
    while TRAVERSED-DISTANCE(look-up-table[index]) ≤ d_t + D_t
      distance←LOCAL-DISTANCE(look-up-table[index])
      horizontal-distance←ABS(cos v)
      vertical-distance←ABS(sin v)
      if horizontal direction is positive then
        point←x(v) + horizontal-distance
      if vertical direction is positive then
        point←y(v) + vertical-distance
      mapped-prototype←point
      index←index + 1
      end
  end
return (mapped-prototype)
```

Figure 4-10 Relative interpolation of a prototype.

4.3.2 Destructive interpolation

After an interpolation process, the unknown can be matched against all the prototypes in the database. Because the data points do not exactly match, the optimal time warping path must be found in order to find the minimum distance between the unknown and the prototype. Since dynamic time warping of two unknown shapes S_1 and S_2 traverses a matrix with i and j data points, the complexity level is $O(ij)$.

However, if the prototype contains the same number of data points as the unknown, every point in the unknown can be matched against every corresponding point in the prototype, thus eliminating the need for

dynamic time warping and reducing the complexity level in the recognition process to $O(i)$.

If the unknown contains enough data points, this can be achieved by simply interpolating the relative data points in the unknown into the prototype and removing the original anchor points from the prototype. The prototype's geometrical shape will probably deform, if the sample rate of the pen is high enough the effect of the deformation can be neglected. Interpolation can be used to increase the sample rate. This is useful for occasions when the user draws a long straight line. Such operations are known to be faster (Isokoski 2001) and thus are more likely to have a lower point data resolution compared to the distance. An example of destructive relative interpolation is shown in Figure 4-11. In a way the unknown acts like a filter on the prototype.

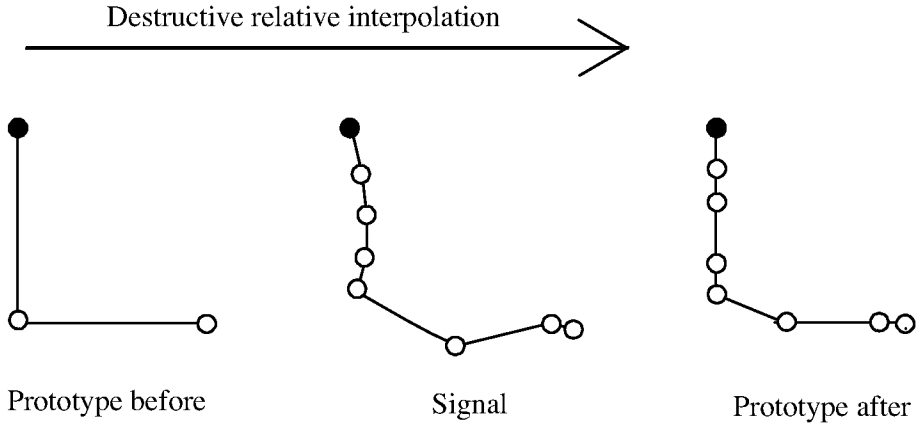


Figure 4-11 Destructive relative interpolation.

4.4 Recognition Engine Design

This section will describe the implemented recognition engine. It has capacity for recognizing both simple geometrical shapes, i.e. shapes that are defined in a small series of data points, and shapes that may contain an unlimited amount of data points.

4.4.1 Data Signal to Prototype Selection

A data signal is received in the form of ordered point data from an input device. The data signal is preprocessed and undergoes a filtering process where duplicate points, and points which do not fulfill certain criterions are removed. The thresholds are user configurable. The unknown is normalized in location and scale. Normalization of location is carried out

by moving the minimal bounding box center of the unknown to the origin of the coordinate system. Normalization of scale is performed by resizing the largest side of the minimal bounding box of the unknown to a predetermined size while retaining the aspect ratio of the shape.

Next, a minimum cost $c_{\min}$ is initialized to ∞, and the system picks a prototype in the database and computes the distance d between the unknown and the selected prototype. If $d \leq c_{\min}$, $c_{\min} \leftarrow d$.

For prototypes consisting of simple geometrical shapes, data points of the unknown are interpolated relatively onto the prototype and the anchor points on the prototype are removed, making the unknown and the prototype containing an equal amount of data points. The distance is computed by adding up the Euclidean distances between each corresponding point.

Non-simple geometrical prototypes are compared to the unknown using elastic matching with the normalized point-to-point technique and the cost function set to the Euclidean distance as the metric. After every prototype has been compared, the prototype whose distance $d = c_{\min}$ is seen as the best match.

An overview of the process is shown in Figure 4-12.

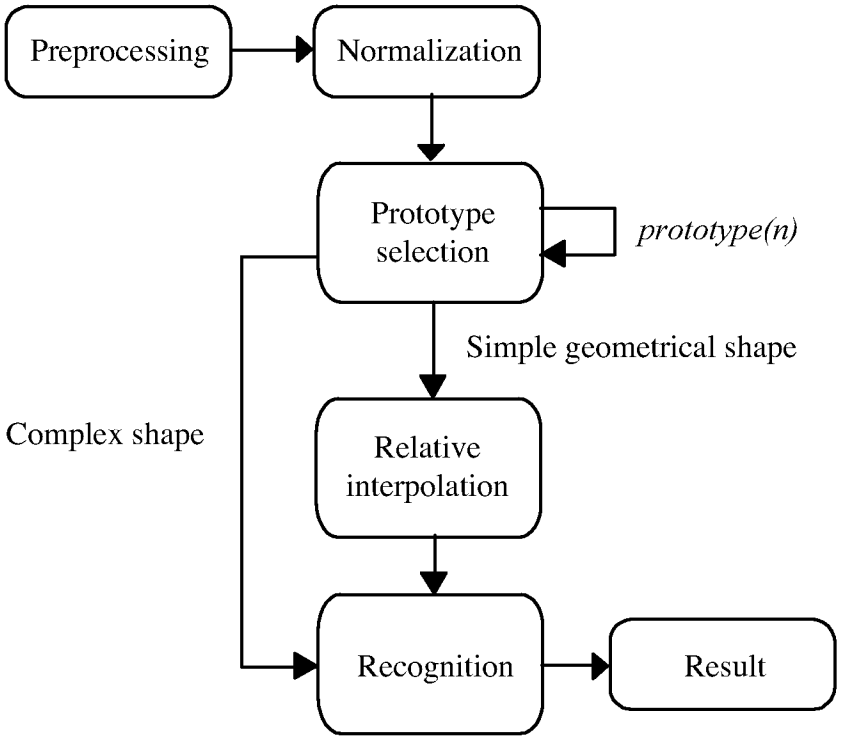


Figure 4-12 UML chart of the recognition process.

Although the process of recognizing simple shapes requires more steps, the individual steps are simpler. The recognition process of arbitrary shapes requires dynamic time warping through a matrix, which is as big as the number of data points in the sample multiplied with the number of data points in the prototype; while the recognition process for simpler shapes simply traverses a list which is as long as the number of data points in the sample or the prototype.

It is worth noting that all shapes in the SHARK system are simple shapes. However, to enable the recognizer to efficiently recognize any shape, like circles and ellipses, the recognizer was built for this dual-mode processing.

How many points does a shape need to have to be defined as a simple one or a complex one? Both kinds of shapes can be recognized using either technique, the question is more one of performance. No formal evaluation of when one of the techniques is more applicable than the other has been made. Recognizing user-defined shapes without heavy filtering however, most probably would put the simple shape recognition technique behind the other.

Chapter 5
User Study

There are many interesting questions regarding the SHARK system. Since the usage and training aspects of the ATOMIK keyboard layout have been studied before (Zhai 2001), perhaps the most basic question is if users can learn the keyboard-originated gestures at all. The second question is how effectively the users can be trained and what performance can be expected after a certain period of training?

In an attempt to answer the above questions a training experiment was conducted. Given the success of the expanding rehearsal interval (ERI) in previous training experiments in stylus soft keyboarding (Zhai et al 2002) it was determined that it might be viable to adapt a similar strategy.

The ERI algorithm adapts the training after the user performance. If the user does well on a word, the rehearsal interval for the word is increased, if he/ she fails to remember the rehearsal time stays the same (or is decreased, depending on the implementation) (Landauer & Bjork 1978, in Zhai et al 2002). The idea is that the algorithm should train the user on the words he/ she remembers the least at current time, rather than just rehearsing all words at an equal interval regardless of which words the user actually needs to practice more on. As Zhai et al (2002) notes, this is similar to the strategy applied when learning foreign words. You typically train on a word more in the beginning, and when you know the word better you rehearse it less, gradually decreasing the rehearsal interval.

5.1 Task

The task was to learn as many shorthand gestures for words as possible in four training sessions of approximately 45 minutes each. The fifth session was a final test, where the users were tested against all the words they practiced. This test was conducted one day after each training session.

5.2 Apparatus

The training is carried out by a computer application implementing the SHARK system. The recognition engine used elastic matching and the normalized point-to-point distance with the Euclidean distance between the data points as the distance metric between two strokes. 100 words were converted to prototypes and loaded into the recognition engine database. The experiments were carried out on an Athlon™ XP machine running Microsoft Windows® version 5.0. The application was programmed in Java™, compiled into bytecode using IBM Jikes™ and executed on the official Sun Java Virtual Machine version 1.4.0. The application used the standard Microsoft Windows® win32 look and feel (figure 9.1). The recognition results were returned in real-time to the user. Pen input was sampled using a Wacom tablet model ET-0405-U and the pen traces were echoed on a special designated canvas on the screen with a resolution of 400 x 300 pixels. The Wacom tablet had an active area size of 334 x 258 mm and a resolution of 2540 lpmm. Max data rate was 200 pps, it was attached to the computer using the USB interface. The Wacom tablet's active area was calibrated to map the entire screen. The screen resolution was 1280 x 1024 pixels on a 22" Mitsubishi CRT monitor with a refresh rate of 120 Hz vertically. Signal processing techniques were used to reduce the data load. The filtering algorithm explained in 4.2.1.1 was used with the `max_angle` parameter set to $\frac{\pi}{4}$ radians and the `min_distance` set to 4 pixels. The unknown was normalized in shape and location. The unknown were scaled down retaining aspect ratio by resizing the largest side of the minimal bounding box to 100 pixels. The geometrical center of the minimal bounding box was then moved to the origin of the coordinate system.

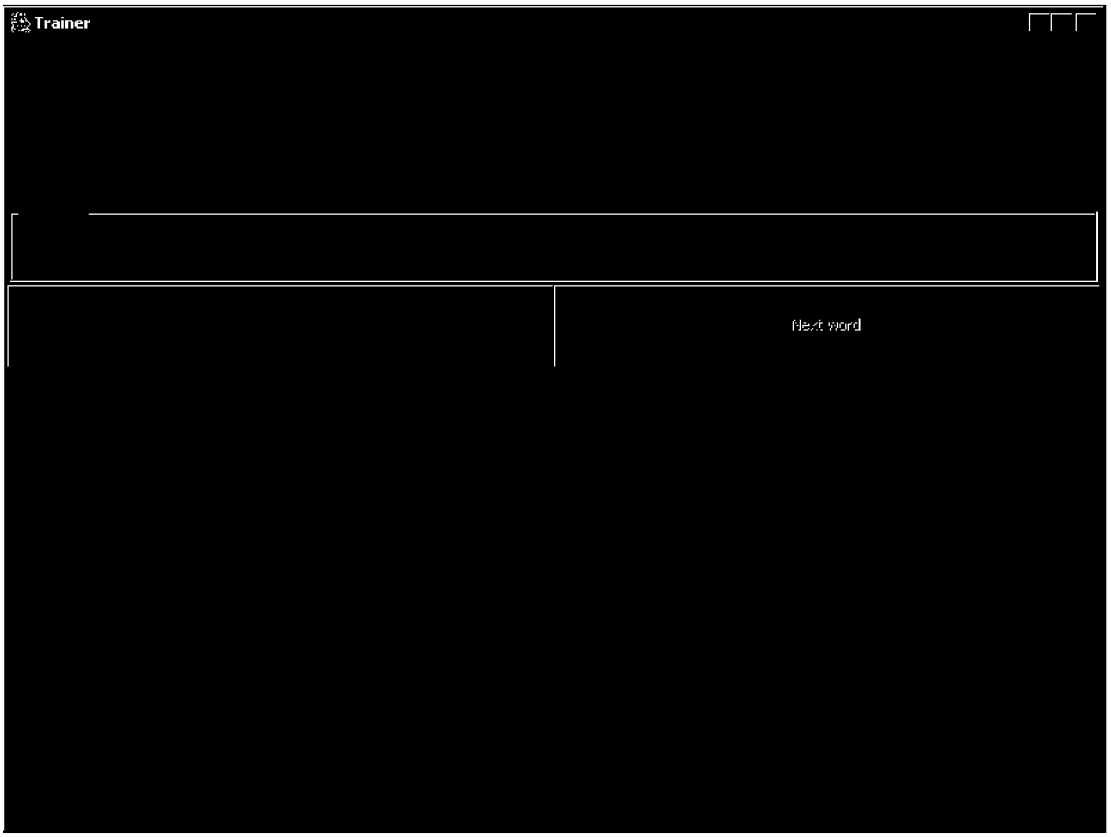


Figure 5-1 Screenshot of the training application.

When the user pressed the stylus against the Wacom tablet and the screen cursor was located in a dedicated canvas on the computer screen, the pen strokes were echoed to the screen[1] as interconnected blue lines with a weight of approximately two pixels. As soon as the user let the pen leave the Wacom tablet surface, the system searched among the prototypes for the best candidate and printed out the corresponding word on a label inside a 'Last entry' widget on the screen.

Since this was a strict memory test, no disambiguation was used. Because of this, the words needed to be properly separated in prototype space. The words used in the experiment were selected by hand. 50 of the 100 words used in the experiment were selected from the top 100 most common words in the English language. The next 50 words used, were selected from the top 300 most common words in the English language. The top-frequency words in the English language were collected from the British National Corpus (BNC).

[1] Also referred to as *electronic ink* (Tappert 1990).

5.3 Subjects

Six paid subjects were engaged in the study. They were all recruited from the Linköping University campus. Four of them were female and two were male. A prize was awarded for the subject who remembered most shorthand gestures at the final test.

5.4 Procedure

Every session, except the first and the last one, began with a short test of the words rehearsed in the previous training sessions. The first session was a training-session only and the last session was only a final test of all the trained words.

5.4.1 Training Mode

The system works by handling two different lists of words:

1. The *word list* contains all the words the user should learn.
2. The *rehearse list* contains all words that the user is currently training.

The training algorithm works as follows:

1. If the rehearse list is empty, pick a new word from the word list and initialize it with a rehearse interval of 30 seconds and put it in the rehearse list.
2. Pick the word from the rehearse list that have the earliest rehearse interval.
3. If the rehearse interval is below 30 s present the word to the user. This is now the practice word.
4. If the rehearse interval is over 30 s, pick a new word from the word list, initialize it's rehears interval to 30 s and go to 2. If the word list is empty, pick the word from the rehearse list with the lowest rehearse interval.
5. If the user remembers the practice word, insert it into the rehearse list with a doubled rehearse interval, else insert the practice word into the wordlist with an unaltered rehearse interval and go to 2.

The training starts with the application selecting a word according to the training algorithm. The word is presented to the user, and the user is asked to write the corresponding word. If the user is unable to write the word, either because he/ she haven't seen the word or doesn't remember

it, the user may tap a button *Show keyboard* to pop up a image of the ATOMIK soft keyboard. In addition, the word that was requested is traced with a series of dashed lines along the corresponding keys (figure 9.2). If the user pops up the soft keyboard, the word is counted as a miss. Alternatively, the user may attempt to write the shape. If the shape is successfully identified as the requested word, the word is counted as a hit; otherwise, the word is counted as a miss. A message was printed out reading *CORRECT* if the user wrote the correct word and *INCORRECT* otherwise. The feedback message was colorcoded (green/ red respectively).

As soon as the user has either pressed 'Show keyboard' or attempted to write the shape, the system goes into *idle* mode. In idle mode, the user may produce any stroke he/ she wants but the strokes are not taken into consideration. This lets the user practice a new word thoroughly before proceeding. When the user feels ready, he/ she taps the *Next word* button and the procedure is repeated. Each training session lasted 45 minutes, the subjects were asked to take a five-minute break at the middle of the session. At the end of the session the rehearse list and the word list states were saved to disk.

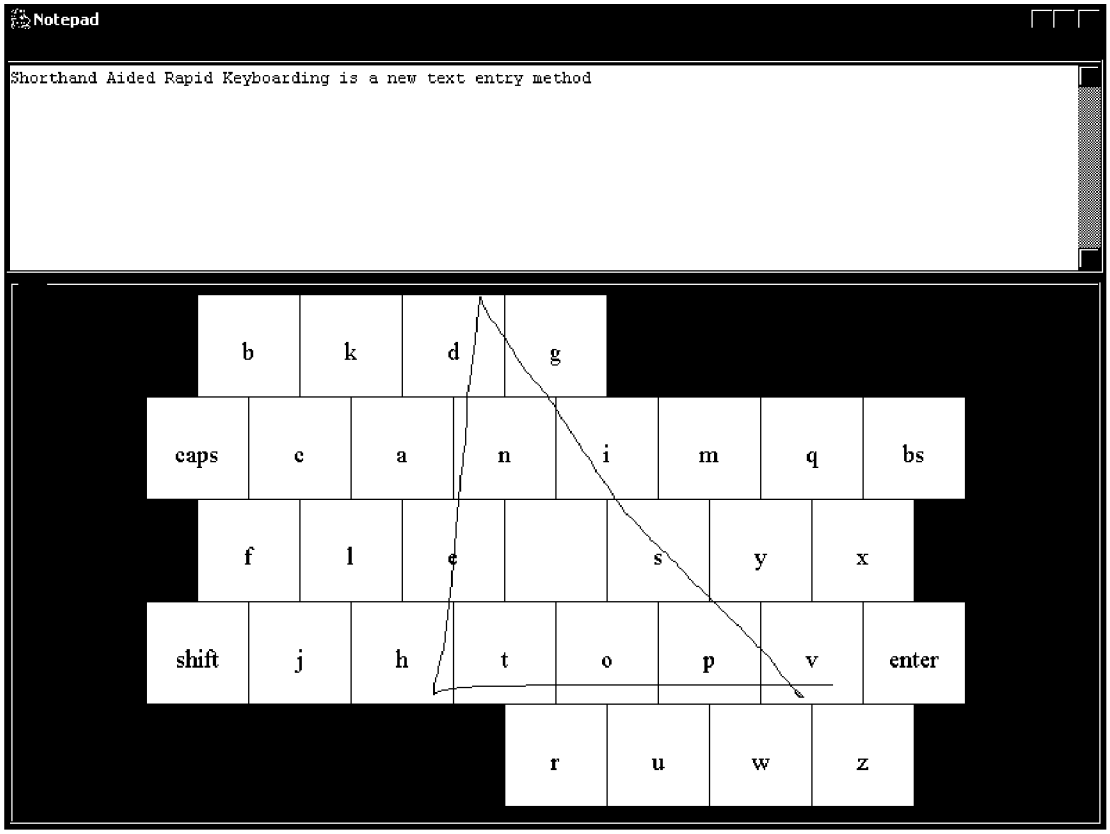


Figure 5-2 *Show Keyboard* operation.

5.4.2 Testing Mode

In testing mode, the application scans the rehearse list file of the user and creates a test list with the words in a randomized order. In other words, the test list consists of all words the user has previously practiced. Each word in the test list is presented to the subject, which is asked to write the corresponding shape. If the user fails the first try, a second chance is given. If the user fails the second time, the next word in the test list is presented to the user. In test mode, the *Show keyboard* button is inaccessible. This is repeated for all words in the test list. The second chance is given because the subjects may indeed remember the shape[2], but they may remember it slightly incorrect or are temporarily imprecise with pen. Since this is a memory test, we felt it would be interesting to see how many words the user remembered if given a second chance. The amount of shape misinterpreted by the system would also serve as an evaluation of the recognizer.

5.5 Results

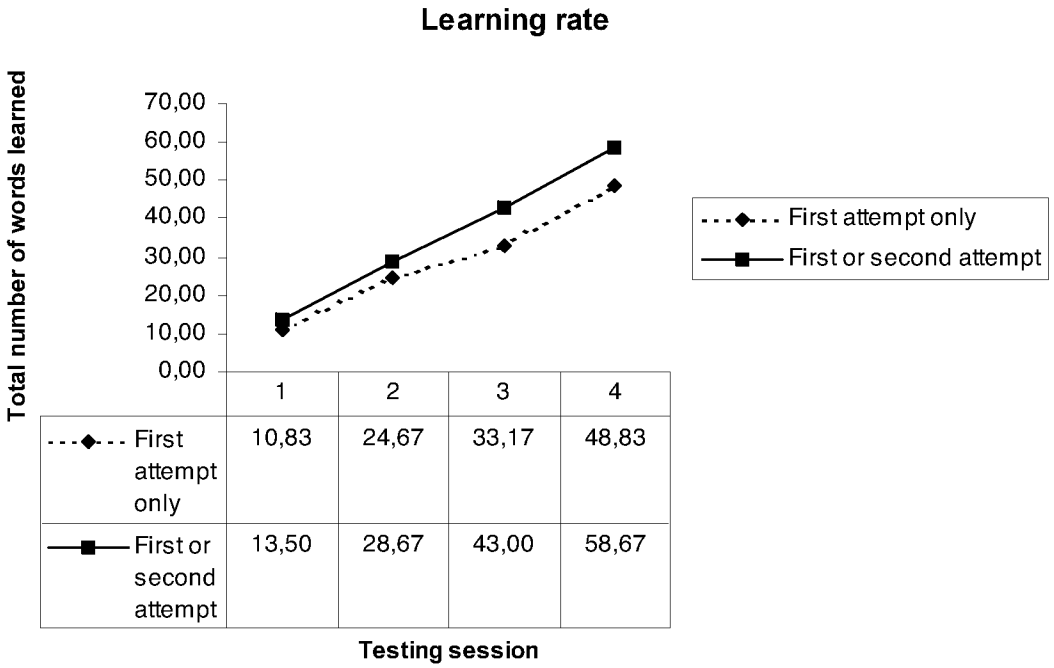
5.5.1 Memory Performance

The results showed that the subjects were able to recall 58.67 words at average at the final testing. This number includes words that the subjects managed to complete when giving a second chance. If we exclude this, the rate falls to 48.83. The lower rate is probably partly due to the recognition being picky about certain gestures, and partly that the subjects make an erroneous gesture and realize their mistake, e.g. they mistakenly draw a mirror image of the gesture for example.

The results indicate that a user indeed is able to learn and reproduce the gestures. Interviews with the subjects showed that they appreciated the expanding rehearse interval training since it made the training "less boring". Opinions from the subjects suggests that the rehearse interval could be tweaked though. We used 2 as the multiply constant, but some subjects complained that they felt that newly introduced words disappeared to quickly in the beginning. Another multiply constant could perhaps improve the learning. However, some subjects said that they liked the expanding rehearsal the way it is, suggesting that the multiply constant maybe should be set on an individual basis.

---

[2] This is only true to some extent. It is vital to understand the implication of the high sample rate from the Wacom tablet, the normalization in location and scale and the properties of the normalized point-to-point distance. Given the sufficient prototype space, a subject will *always* get a word correct given he/ she writes the shape proportionally correct. However, the allowed "sloppiness space" varies among the words.



	1	2	3	4
First attempt only	10,83	24,67	33,17	48,83
First or second attempt	13,50	28,67	43,00	58,67

Figure 5-3 Plot of words mastered between sessions.

The progress between the training sessions was almost constant. The amount of progress between the third testing and the fourth were as high as before, and even higher for some subjects. In fact, the mean progress among the subjects had its peek the very last session. This indicates that the training didn't hit the ceiling, and that there is a great chance that further testing will show that it is possible to learn the 100 most common words as gestures in reasonable time.

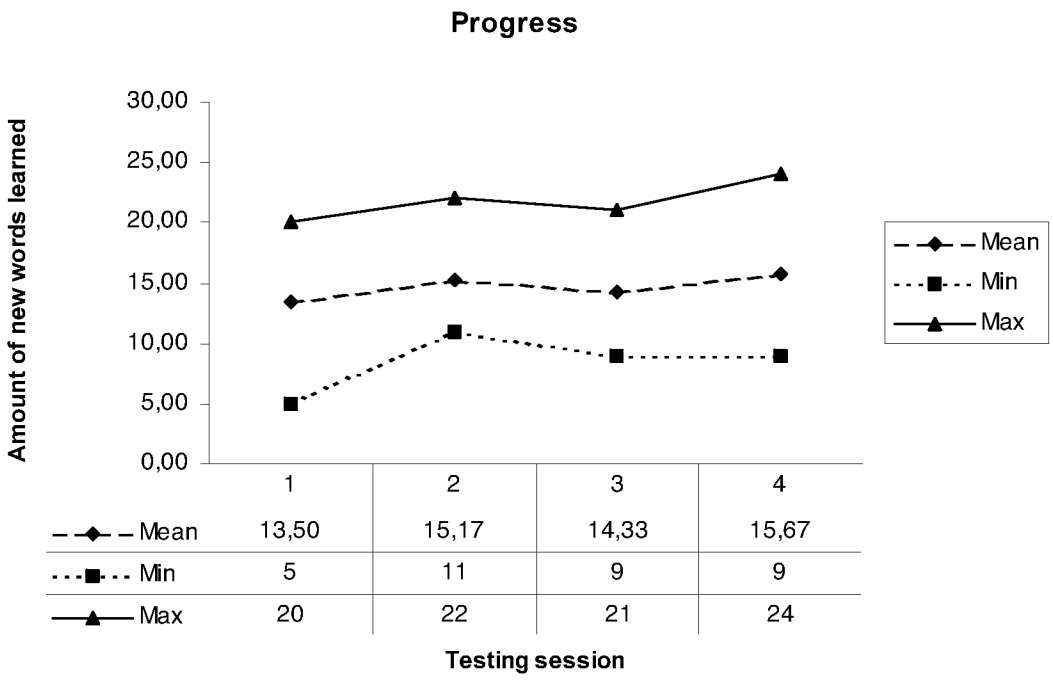


Figure 5-4 Plot of the progress between the testing sessions.

5.5.2 Speed

Since we conducted a strict memory test, we didn't tell the participants in the study to complete the strokes as fast as they could. However, some figures around the speed of the strokes might give an idea of the capacity of the text entry technique.

The stroke times were logged by time stamping the time from the stylus hit the Wacom tablet to the time it left the surface. The fastest word that all subjects practiced was 'him' with a stroke completion times ranging from 240 to 1152 ms and a mean of 570 ms. When looking at the slowest words, it was noted that several strokes lasted up to about 9 – 16 seconds. However, this is probably due to the habit of the subjects to begin a stroke and after producing a line starting to think from where to continue next. We are therefore careful not to interpret too much from the collected data set. The speed data for individual words can be found in Appendix F.

5.5.3 User Attitudes

Judging from the interviews and the questionnaires people didn't seem to find the system tedious. Some users stated that they really needed a

"strategy" in order to learn the words. With strategy we believe they mean some kind of heuristic, one subject for instance constantly asked the research leader if he/ she should try to learn the keyboard layout or learn the gestures. However, these people had similar performance as the others. It is possible that the artificial testing situation added to the confusion. The participants were for example only able to produce gestures; the soft keyboard was locked during the experiments. Hence the soft keyboard might not make much sense for them.

All subjects perceived the task of reproducing gestures as "kind of fun" in the beginning. Naturally this excitement declined during time, however nobody reported the sessions as tedious or plain boring. We believe these kinds of user attitudes are important since some initial training time will probably be unavoidable in order to use the system efficiently.

In the final questionnaire, participants rated 0.6 on the scale of (–3 frustrating, +3 satisfying); 1.83 on (–3 dull, +3 stimulating); 0.6 on (–3 difficult, +3 easy) to use; 0.6 on (–3 difficult, +3 easy) to learn. To the question "If such a method is made available for practical use, would you learn it?", their answer averaged 1.6 on the scale of (–3 definitely no, +3 definitely yes). When asked if they would use it in a list of situations, 0/6 participant choose "not at all", 2/6 answered "Yes, when a physical keyboard is not available" and 4/6 "Yes, to replace keyboard typing sometimes", and 0/6 "Yes, to replace keyboard typing all the time".

Most subjects found it frustrating that the system was "unfair", i.e. they found the prototype picky about some gestures. Since the prototype was using an earlier recognition algorithm and we put in 100 words when it had mostly been tested with 50 – 60 words, some prototypes were not enough separated in prototype space in order for the algorithm to handle too sloppy handwriting. For real usage an improved recognition algorithm or harsher constraints are probably necessary.

Chapter 6
Discussion and Future Work

There are many interesting aspects of the SHARK system. The construction of a working prototype and the successful user study suggests that a viable product could indeed prove as a successful alternative for rapid pen based text entry.

6.1 Issues

6.1.1 Recognition Problems

The user study showed that the recognition engine was a bit picky about certain gestures. This is partly due to the early version of the recognition engine and partly due to the non-existent disambiguation algorithm due to the nature of the study.

The user study showed that users are very unforgiving about errors that they blame the system on. They get irritated and the text entry "flow" suffers from an abrupt pause. Moreover, previous research has shown that a text entry system must reach an overall reliable accuracy of 97% percent in order for users to accept it (LaLomia 1994, in MacKenzie & Zhang 1997).

The case of the recognition engine returning a different "class" of word than the one the user believed he/ she stroked is interesting. For instance, if the user produced 'them' almost correct, but got returned with 'his' the system selection would appear very illogical from the user's point of view (Figure 6-1). The user probably clearly produced three interconnected vectors but the system returned a word consisting of only two vectors. These tendencies to choose the wrong class of words were reported by two users in the survey, in the "Open comments" field in the questionnaire. This is very interesting case, since a study of how the users

produces the strokes can lead to insights in what aspects of the recognition processes that can be enhanced.

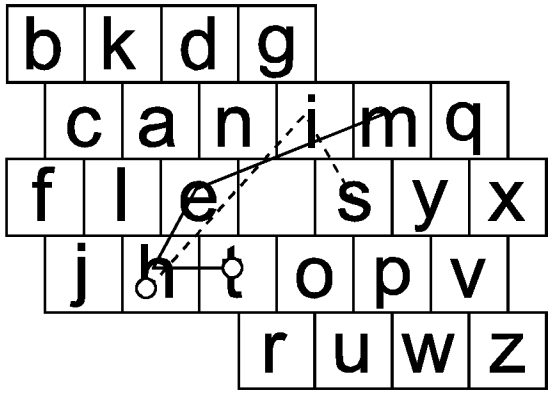


Figure 6-1 'them' and 'his' plotted on ATOMIK.

There are many approaches to resolve the recognition problems. First of all, a location dependent disambiguation would resolve a lot of problems. However, to ease and possibly speed up the text entry of gestures, the recognition system should be very robust and preferably emphasis segments than proportions. Appendix A – Enhancing the Recognition Engine, discusses this subject in more depth.

6.1.2 Partial Location Dependence

Making the soft keyboard partially location dependent has some major advantages for fast disambiguation as discussed in 3.5.2. However, partial location dependence has it's own cost too. Since we are constraining the gestures to an enclosed area we are also forcing the users to do a visual search in order to locate the gesture at an approximate place. This has two implications:

1. True heads-up writing is impossible for some gestures.
2. The prototype space of the gestures expands.

6.1.2.1 Heads-up Writing

We judge heads-up writing as a minor issue, mostly due to the frequent need of closed-loop visual attention anyway since statistically the user is required to use the soft keyboard about 55% of the time (implication of Zipf's law, 3.4.1). Still, heads-up writing is a powerful feature of text entry: allowing the user to relax the visual attention from the device and use it on another source, like a business presentation, a lecture, etc is

clearly beneficial. It would be an advantage for SHARK supporting completely open-loop operation by the user. This is indeed mostly the case since most gestures are not ambiguous and can be stroked in an open-loop fashion. If we study some gestures that would need a disambiguation procedure we discover that they are widely separated even if they follow the partial location dependency paradigm. If we take the words 'an' and 'can' for example, the user only need to put down the pen on one side of the letter-pairs ('c'; 'a') and ('a'; 'n'). Hence, although theoretically some amount of visual attention is needed, in practice this might not be the case. This is evident when studying Figure 6-2.

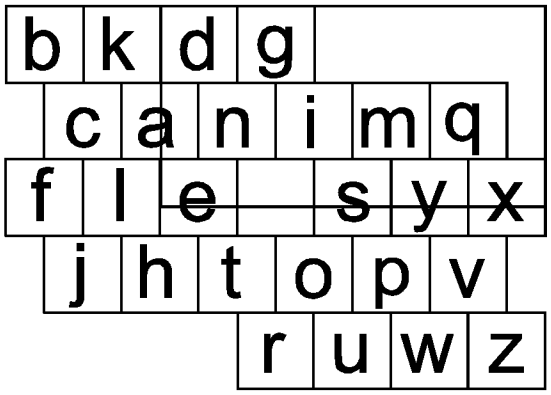


Figure 6-2 Location dependent area for stroking 'an'.

6.1.2.2 Extended Prototype Space

An advantage of partial location dependence is the extended prototype space. A system could be designed to take advantage of the location dependence and emphasis location when the stroke is judged as very sloppy and uncertain to recognize. In other words the system could be designed with some degree of flexibility: a user may stroke somewhat proper strokes in an open-loop fashion for most strokes, or draw more sloppy strokes in a more location dependent way. This is a huge advantage when considering the soft keyboard mode. Given a user has a fairly good idea of where to tap characters on the keyboard he/ she may just draw a very quick sloppy gesture at the approximate location. Since no gesture matched reliably the system checks the location of the strokes and picks the word whose shape is geometrical closest to the stroke. Such an operation could be user controlled with a parameter setting the weight for location dependency.

6.2 User Perspectives

6.2.1 Learning

As shown in the user study the subjects managed to learn 15 new words per session. Since the progress was rather constant between the sessions, the subjects in the user study showing no sign to hit the ceiling, there is a promising possibility to learn close to 100 words or more. Shorthand literature indicates that it is indeed possible but the question is about how fast. As mention in previous research papers, users typically do not want to spend a lot of time learning a new text entry system.

The issue of learning the soft keyboard is also interesting. Previous learning experiments on learning the ATOMIK soft keyboard suggests that about 10 sessions of training are needed to reach high speeds (Zhai et al 2001). One could hope for synergy effects between learning the gestures and learning how to use the keyboard layout, since they share several obvious properties.

6.2.2 Supporting the Novice and the Expert

As suggested by MacKenzie (1997) and several others, the novice users first experience with a new text entry system is important. A nice aspect with the SHARK system compared to other gesture recognition systems is the keyboard fallback mechanism. If a user cannot remember a stroke or find it too difficult, he/ she can always tap the word. Moreover, if the user has forgotten a specific gesture he/ she does not need to find a reference card to look at, the very shape is shown in the form of the spatial relationship between the keys of the soft keyboard.

An expert, on the other hand, is better supported using the SHARK system than using most other pen based text entry systems. ATOMIK is one of the theoretically fastest soft keyboard layouts available (Zhai et al 2001) and knowledge of the gestures for the most common words probably enhances text entry performance although this remains to be proven.

6.3 Future Research

6.3.1 Smart concatenation of text units

The SHARK system is currently at an early prototype stage and only primitive integration to an actual text entry application has been made.

Currently the system outputs words with a space character automatically in between independently of the current context in the sentence. Future enhancements lies in making the text entry system more intelligent, allowing seamless SHARK gesture text entry, paying attention to punctuation and common parts of words that may be stroked individually like 'ing' in 'stroking'.

6.3.2 Learning Progress

Since the user study wasn't conducted long enough to show how many gestures the users could learn, a longitudinal learning experiment would certainly be interesting. There are many aspects in the learning process like optimization of learning progress, max limit of gestures to be remembered and the amount of loss of gestures if they are not rehearsed or used actively.

6.3.3 Actual User Text Entry Performance Data

No study on text entry speed using the SHARK system has been conducted. There are reasons, outlined in this thesis; to believe that it should perform competively with other pen based state-of-the-art text entry systems (in electronic devices) currently in use. However since no actual user data exists it is hard to say anything about the average user speed for the system.

6.3.4 Do Users Take Advantage of the Gestures?

How should a user know which of the words exists as gestures? Intuitively a user may guess that certain very common words exists like *if*, *and*, *the*, etc. However, some sort of "tip" function could be implemented into the software. When the user has tapped a word that exists a gesture the corresponding keys may display an animation of the shorthand gesture trajectory for example to indicate that there is a more comfortable way of entering this word. Since the most common words by definition are likely to occur frequently such a feature may be a form of training while the text entry system is in actual use. This could be interesting, since people likely want to get the job done than instantly being forced to train on the system for the next few hours.

6.3.5 Arbitrary Shorthand Gestures

The recognition system developed has capacity for recognizing not only SHARK shorthand gestures, but also arbitrary shapes. This opens up the

possibility to allow completely user defined shorthand gestures, like for example spirals, arrows and circles.

6.3.6 Similarity of Shapes

A few very interesting phenomena were discovered when training the users on the gestures. The users tended to confuse simple shapes consisting of one or two strokes with each other. An interesting aspect of this confusion is that the subjects often produced a complete mirror shape of the gesture in question, even though no such gesture existed. Other effects noted were the difficulty to produce proportionally correct shapes if the angles were too sharp. Shape similarity of pen gestures is an interesting subject that has gotten some attention lately, e.g. Long, Landay, Rowe & Michiels (2000). There are clearly some learning effects noted with aspect of the shapes. For instance many subjects claimed they had more difficulty learning the simple shapes than the more complex ones. Indeed, the mirror shape phenomenon almost exclusively occurred for shapes connected with 1 – 3 segments.

6.4 Conclusions

In this thesis we have shown that

1. It is possible to construct a shorthand aided soft keyboard that is useable in practice.
2. A user study indicates that users are able to learn about 15 new gestures per 40 minutes training session. Additionally the ceiling does not seem to have been hit.

The developed system, SHARK, appears to have several great advantages compared to other pen based text entry methods, the most important being a potential speed boost compared to stylus touch typing on an optimized keyboard layout, which are among the fastest text entry systems for mobile devices as of today (2002).

We came to the conclusion that future work and experiments are needed to conclude if this is a better text entry method than ones already present or not. In addition we have suggested several ways and methods to research and develop the system to further enhance its feasibility within the field of mobile text entry.

References

Accot, J., Zhai, S. 1997. "Beyound Fitts' Law: Models for Trajectory-Based HCI tasks". *Proceedings of CHI '97*, ACM: pp. 295 – 302.

Accot, J., Zhai, S. 1999. "Performance Evaluation of Input Devices in Trajectory-based Tasks: An Application of The Steering Law". *Proceedings of CHI '99*. ACM: pp. 466 – 472.

Aksela, M. 2000. *Handwritten Character Recognition: A Palm-top implementation and Adaptive Committee Experiments*. Master's Thesis, Helsinki University of Technology.

Arakawa, H. 1983. "On-line Recognition of Handwritten Characters – Alphanumerics, Hiragana, Katakana, Kanji". *Pattern Recognition*, 16(1): 9 – 16.

Beigi, H. S. M. 1993. "An Overview of Handwriting Recognition". *Proceedings of the 1st Annual Conference on Technological Advancements in Developing countries*, Columbia University, New York, July 24 – 25: pp. 30 – 46.

Blickenstorfer, C. H. 1995. "Graffiti: Wow!". *Pen Computing*, January: pp. 30 – 31.

Christopher, T. W., Thiruvathukal, G. K. 2000. *High-Performance Java Platform Computing*. Prentice Hall: New Jersey.

Cormie, D. 2002. *The ARM11™ Microarchitecture*. URL: http://www.arm.com/support/White_Papers?OpenDocument/. (Verified June 1, 2002).

Dougherty, E. R. 1999. *Random Processes for Image and Signal Processing*. IEEE Press, New York.

Goldberg, D., Richardson, C. 1993. "Touch-typing with a stylus". *Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems*, ACM: pp. 80 – 87.

Goldstein, M., Book, R., Alsiö, G., Tessa, S. 1999. "Non-Keyboard QWERTY Touch Typing: A Portable Input Interface For The Mobile User". *Proceedings of the CHI '99 conference on Human Factors in Computing Systems*, ACM: pp. 32- 39.

Gosling, J., McGilton, H. 1996. *The Java Language Environment – A White Paper*. URL: http://java.sun.com/docs/white/langenv/. (Verified May 19, 2002).

Guernsey, L. 2000. "Playing Taps on the Cell Phone". *New York Times*, October 12th 2000.

Hunter, M., Zhai, S., Smith, B. A. 2000. "Physics-based Graphical Keyboard Design". *CHI '2000,* April 1 – 6, The Hauge, The Netherlands.

Isokoski, P., Raisamo, R. 2000. "Device Independant Text Input: A Rationale and an Example". *Proceedings of the Working Conference on Advanced Visual Interfaces*, ACM: pp. 76 – 83.

Isokoski, P. 2001. "Model for Unistroke Writing Time". *CHI Letters: Human Factorns in Computing Systems*, 3(1): 357 – 364.

Jurafsky, D., Martin, J. H. 2000. *Speech and Language Processing – An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice-Hall Inc., New Jersey.

Karat, C.-M., Halverson, C., Horn, D., Karat, J. "Patterns of Entry and Correction in Large Vocabulary Continuous Speech Recognition Systems". 1999. *Proceedings of CHI '99, ACM Conference on Human Factors in Computing Systems,* Pittsburgh, PA, USA: pp. 568 – 574.

Landauer, T. K., Bjork, R. A. 1978. *Optimum rehearsal patterns and name learning, in practical aspects of memory.* P. E. M. Gruneberg, M. M. & Sykes, R. N. Editor. Academic Press: London.

Leech, G., Rayson, P., Wilson, A. 2001. *Word Frequencies in Written and Spoken English: based on the British National Corpus*. Longman: London.

Leedhamn, C. G., Downton, A. C., Brooks, C. P., Newell, A. F. 1984. "On-line Acquisition of Pitman's Handwritten Shorthand as Means of Rapid Data Entry". *Proceedings of IFIP INTERACT '84*, Human-Computer Interaction: pp. 145 – 150.

Leedhamn, C. G., Downton, A. C. 1986. "On-line Recognition of Pitman's Handwritten Shorthand – An Evaluation of Potential". *International Journal of Man-Machine Studies*. 24(4): 375 – 393.

Long, A. C. Jr., Landay, J. A., Rowe, L. A., Michiels, J. 2000. "Visual Similarity of Pen Gestures". *CHI 2000, ACM Conference on Human Factors in Computing Systems*, CHI Letters, 2(1): 360 – 367.

Looney, C. G. 1997. *Pattern Recognition Using Neural Networks – Theory and Algorithms for Engineers and Scientists*. Oxford University Press, New York.

Mankoff, J., Abowd G. D. 1998. "Cirrin: A word-level unistroke keyboard for pen input". *Proceedings of UIST '98*. Technical note: pp. 213 – 214.

MacKenzie, I. S., Nonnecke, B., Riddersma, S., McQueen, C., Meltz, M. 1994. "Alphanumeric entry on pen-based computers". *International Journal of Human-Computer Studies*, 41(5): 775 – 792.

MacKenzie, I. S., Zhang, X. S. 1997. "The immediate usability of Graffiti". *Proceedings of Graphics Interface '97*: pp. 129 – 137.

MacKenzie, I. S., Zhang, X. S. 1999. "The Design and Evaluation of a High-Performance Soft Keyboard". *Proceedings of the ACM Conference on Human Factors in Computing Systems – CHI '99*: pp. 25 – 31.

MacKenzie, I. S., Kober, H., Smith, D., Jones, T., Skepner, E. 2001. "LetterWise: Prefix-based Disambiguation for Mobile Text Input". *Proceedings of the ACM Symposium on User Interface Software and Technology – UIST 2000*: pp. 111 – 120.

MacKenzie, I. S., Zhang, X. S. 2001. "An Empirical Investigation of the Novice Experience With Soft Keyboards". *Behaviour & Information Technology*, 20: pp. 411 – 418.

MacKenzie, I. S., Soukoreff, R. W. 2002. "A Model of Two-Thumb Text Entry". *Proceedings of Graphics Interface 2002*: pp. 117 – 124.

Nationalencyklopedin. 1995. Bra böckers förlag: Höganäs. (Swedish dictionary)

Oxford Encyclopedia 1998. Oxford University Press.

Pavlidis, I., Singh, R., Papanikolopoulos, N. P. 1996. "Recognition of On-line Handwritten Patterns Through Shape Metamorphosis". *Proceedings of the 13th International Conference on Pattern Recognition*, Vol. 3: pp. 18 – 22.

Perlin, K. 1998. "Quikwriting: Continuous Stylus-based Text Entry". *ACM UIST '98*, Technical Sketch: pp. 215 – 216.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenson, W. 1991. *Object-oriented Modeling and Design*. Prentice Hall: New Jersey.

Scattolin, P. 1995. *Recognition of Handwritten Numerals Using Elastic Matching*. Master's Thesis. Department of Computer Science, Concordia University: Montréal.

Silfverberg, M., MacKenzie, I. S., Korhonen, P. 2000. "Predicting Text Entry Speed on Mobile Phones". *Proceedings of the ACM Conference on Human Factors in Computing Systems – CHI 2000*: pp. 9 – 16.

Tappert, C. C. 1982. "Cursive Script Recognition by Elastic Matching". *IBM Journal of Research & Development*, 26(6): 765 – 771.

Tappert, C. C., Suen, C. Y., Wakahara, Y. 1990. "The State of the Art in On-Line Handwriting Recognition". *Transactions on Pattern Analysis and Machine Intelligence*, 12(8): 787 – 808.

Venolia, D., Neiberg F. 1994. "T-Cube: A Fast, Self-Disclosing Pen-Based Alphabet". *Proceedings of CHI '94* (Boston, April 24 – 28, 1994), ACM Press: pp. 265 – 270.

Vuori, V. 1999. *Adaptation in on-line recognition of handwriting*. Master's Thesis, Helsinki University of Technology.

Weiser, M. 1999. "The Computer for the 21st Century". *Scientific American*, September 1991: pp. 94 – 104.

Zhai, S., Hunter, M., Smith, B. A. 2000. "The Metropolis Keyboard – An Exploration of Quantitative Techniques for Virtual Keyboard Design". *Proceedings of ACM Symposium on User Interface Software and Technology (UIST 2000)*, November 5 – 8, San Diego, California: pp. 119 – 128.

Zhai, S., Smith, B. A. 2001. "Alphabetically Biased Virtual Keyboards Are Easier to Use – Layout Does Matter". *Extended Abstracts of CHI2001 – ACM Conference on Human Factors in Computing Systems*, Short Talks, Seattle, WA, April 1 – 5: pp. 321 –322.

Zhai, S., Sue, A., Accot, J. 2002. "Movement Model, Hits Distribution and Learning in Virtual Keyboarding". *CHI 2002*, April 20 – 25, 2002, Minneapolis, Minnesota, USA, CHI letters: (4)1: 17 – 24.

Appendices

Appendix A
Enhancing the Recognition Engine

The current implementation fits the purpose of a simple on-line shape recognizer well. However, if we were to implement the SHARK system in a comfortable to use product, the system must be more tolerant to sloppy "handwriting". A more tolerant recognition system would probably allow for faster gesture text entry too, since the prototype space would be greater, hence increasing the area of the steering tunnels (Accot & Zhai 1997; Accot & Zhai 1999).

A.1 Recognition Problems

The current recognition system is able to successfully recognize any shape if the prototypes are sufficiently separated in prototype space. Due to the nature of the scale and location normalization, and the normalized point-to-point distance metric using the Euclidean distance as the cost function, the recognition is very sensitive to the proportions of the stroke as shown in Figure A–1. Although these strokes are very different for a human, the metric would probably not discriminate between them to the extent that is generally desired. The system does not emphasis small structural differences in a shape, a straight line tends to be confused with a shape consisting of a long straight line connected to a short line.

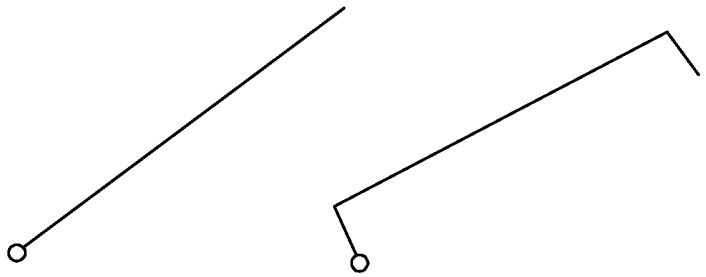


Figure A-1 Weakness of the current implemented algorithm.

A.2 Enhanced Elastic Matching Metrics

There are many ways to utilize elastic matching. Vuori (1999) goes into great detail in exploring the power of DTW and presents matching algorithms that match above the point-to-point level. It is possible, for example, to utilize point-to-line distance, and area based metrics like the Simple-Area distance (SA) and the Kind-of-Area distance (KA) (Vuori 1999) to reach better accuracy. They all, however, suffer from the inability to emphasis the characteristics of the gestures in the SHARK system, i.e. the sequence and *types* of interconnected 2-dimensional vectors.

A.3 Feature Matching

After careful study of the characteristics of the prototypes in the SHARK system, we have gained an interest towards feature matching[1]. Feature matching is a very broad approach, which undertakes many forms, but the basic characteristics of feature matching are to utilize special-purpose algorithms to detect certain specific and characteristic *features* of the unknown shape and the prototypes. The interesting features vary by the geometrical trends of the prototype set.

In the case of the SHARK system it can be noted that the prototypes *always* are interconnected straight lines. A solution to better matching could be to utilize corner and curvature detection algorithms to identify the different segments, as shown in Figure A–2. The shapes are broken down to segments, which are later matched against a pool of segment data collected from the prototypes. In the case of shape *A*, a corner is detected and the shape is interpreted as two line segments. Shapes *B* and *C* demonstrate a major advantage of feature based matching: the two strokes are identified as having different segmentation properties. If we were using the normalized point-to-point distance there is a great risk that shapes *B* and *C* would be interpreted as very similar to each other.

[1] Also known as *stroke coding* (Tappert et al 1990).

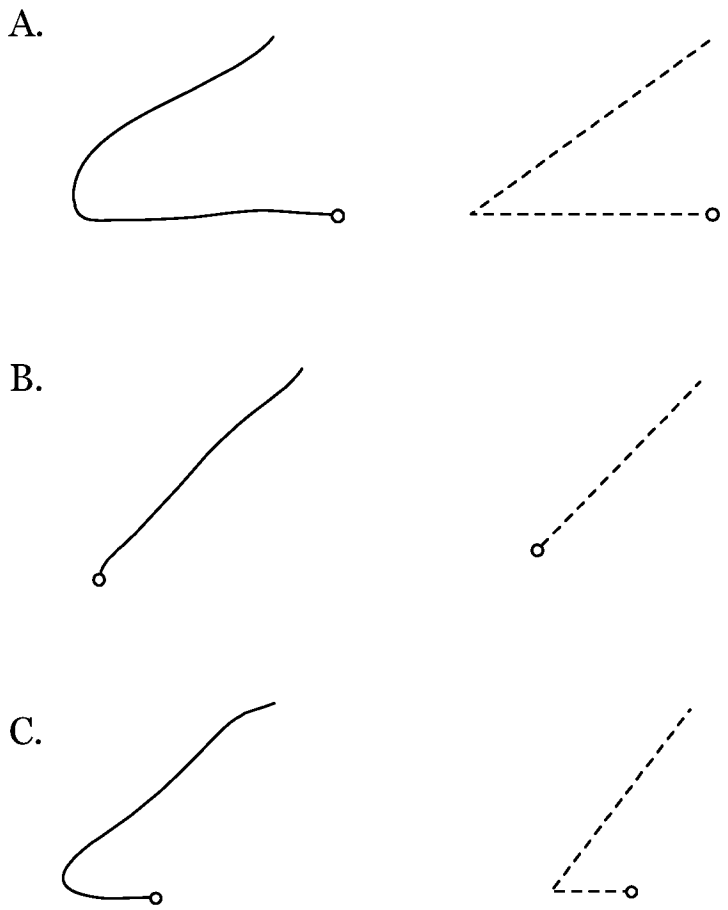


Figure A-2 Feature matching.

However, the situation gets complicated because we typically do not want to force the user to do sharp angles that could slow down the overall time to produce individual strokes. Sophisticated corner detection algorithms have a certain performance cost, but an effective pruning algorithm could probably dramatically reduce the number of prototypes that actually has to be matched. As mentioned before, the characteristics between certain groups of prototypes vary widely, allowing a quick pruning algorithm to filter out prototypes judged as impossible already from the beginning.

A feature matching recognition system has many other advantages as well. The amount of feature data to match against the prototypes would be typically less than the point data comparisons. The costly analysis of curvatures, etc. would only be necessary for the unknown, since the prototypes only have to be examined at initialization time. Feature matching of the unknown against the prototypes can be resolved by the use of tree searches or calculating the Levenshtein distance between two syntactic data strings, which like elastic matching can be solved

efficiently and optimally. Feature matching would probably in the context of the SHARK system's simple geometrical shaped prototypes be a far more reliable matching method than the current implementation. Alternatively a simplistic feature matching algorithm could be used to prune out confusing prototypes already from the beginning of the matching process, and traditional elastic matching could be used to make the final decision.

A.4 Zones

Another approach to improve the performance of the recognition system is to utilize a brute force pruning algorithm that relies on the dynamic stroke information. By dividing the bounding box of the shape into a series of zones and observing the pen traversing these different zones in sequence, the system can filter out impossible alternatives by the help of a pre-computed dictionary of zone sequences among the prototypes (Tappert et al 1990). If a prototype is very characteristic the algorithm may even find the correct candidate directly. In the case of the SHARK system, we believe the zone technique has potential to be a very effective pruning algorithm in order to limit the choices for further processing. However, by observing the characteristics of the gestures in the SHARK system it seems that some particularly troublesome gestures may have the same zone sequence, since they typically aren't well separated in prototype space. An example of the zone technique is shown in Figure A–3. A circle marks the beginning of the strokes. The sequence of the traversed strokes may be used to filter out "impossible" prototypes. Notice the two shapes to the left, which are very different from the perspective of zone pruning. If we applied elastic matching using the normalized point-to-point distance metric, there is a great risk they would be confused.

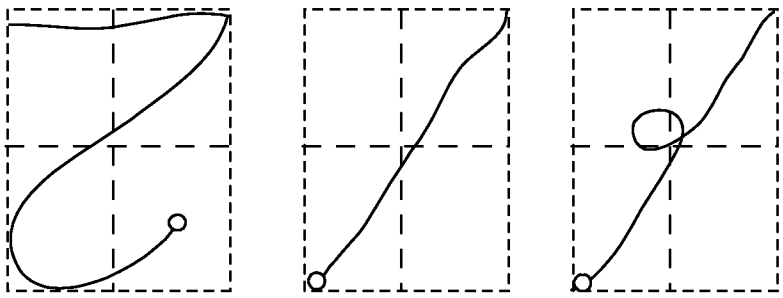


Figure A-3 Zone pruning.

Appendix B
SHARK System Architecture

The SHARK system was developed using Sun Java™ technology (Gosling & Chilton 1996) as the main platform. Although most of the main system is written in Java, the codebase was designed to be easy to translate into C or C ++ code for implementation on the Palm OS® or the Microsoft Windows® PocketPC mobile software platforms.

The system is broadly divided into two parts: the stand-alone recognition engine, which returns the most similar prototype in the database given a signal, and the Java™ Swing GUI soft keyboard component.

B.1 Recognition

The recognition algorithms are defined and explained in chapter 4. Here we will focus on the object structure of the recognition system.

The SHARK system is using a Prototype class as a wrapper for the shape data. This is advantageous since it allows object encapsulation, if we for example wanted to give the prototypes additional features. Because of speed, Prototype objects are converted into an internal data format when they are passed to the recognition engine. Currently the engine uses a double matrix as the internal representation of the prototype database.

B.2 Keyboard GUI Component

A GUI component was developed to feature the actual user interface for shark. It was developed as a JComponent in Java™ Swing, since it would make sense from a research perspective to have it easy plug and playable for various different applications. We currently have a notepad application and a training system implementing the same GUI component. To use SHARK an application simply adds the

KeyboardComponent class to a JFrame as shown in Figure B–1 and Figure B–2.

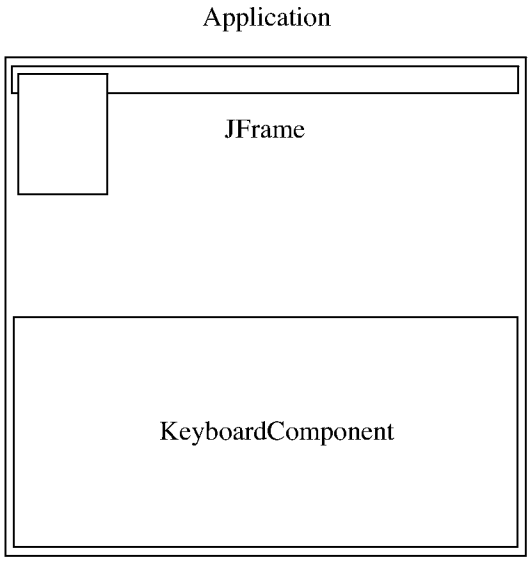


Figure B-1 KeyboardComponent inside a JFrame.

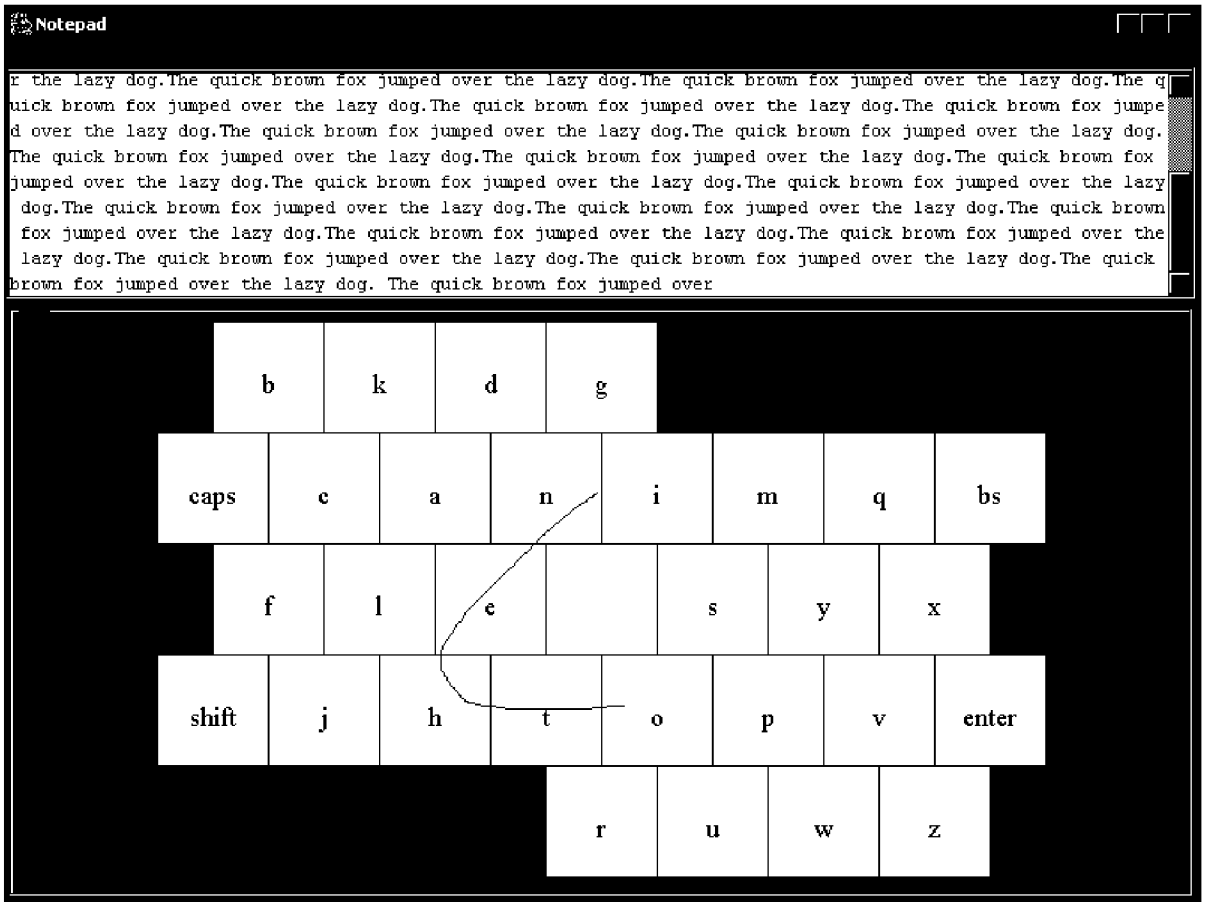


Figure B-2 Notepad GUI.

The keyboard is created using a KeyboardLayout class, which is a container class featuring KeyboardItem objects. Each KeyboardItem contains a unique identification, coordinate data and a key code string that the application will send to the system when the user taps it. The KeyboardLayout class works like a hashtable, the application can query the layout for the unique identification strings and receive the corresponding KeyboardItem. In the SHARK system the ATOMIK layout is build up using a series of KeyboardItem objects for all the letters A – Z. When the system builds the prototypes, it takes a word and queries the layout for the coordinate data of the letters in the word. These coordinates are augmented into a stroke that are fed to the recognition engine as a Prototype object.

B.3 Threading

Since the SHARK GUI component lies within an application, the recognition process in the SHARK system should preferably be threaded. If SHARK was a singly threaded application, the entire user interface would freeze while SHARK is doing something processor intensive, like identifying a gesture from the prototype database.

Therefore, the SHARK system separates the java native GUI thread, the Swing thread, and the recognition engine processing thread. Since there are sometimes the need for a non-threaded naive application due to porting and debugging reasons, two threaded wrapper classes for the GUI component and the recognition engine was developed.

The recognition process and the GUI interface runs in separate threads and need to exchange data in a deterministic matter. This was solved using the producer – consumer design pattern. A mutual exclusive data structure was modeled into a separate object that is used as a bridge between the threads. The shared data structure contains a lock, which is set when an instance is reading or writing the structure and released afterwards. Upon release the shared data structure notifies all instances that the data structure is ready for access (Christopher & Thiruvathukal 2000). Utilizing this technique the thread instances loop in an infinite loop and waits for a notify event from the shared data structure.

The entire system is modeled in Figure B–3.

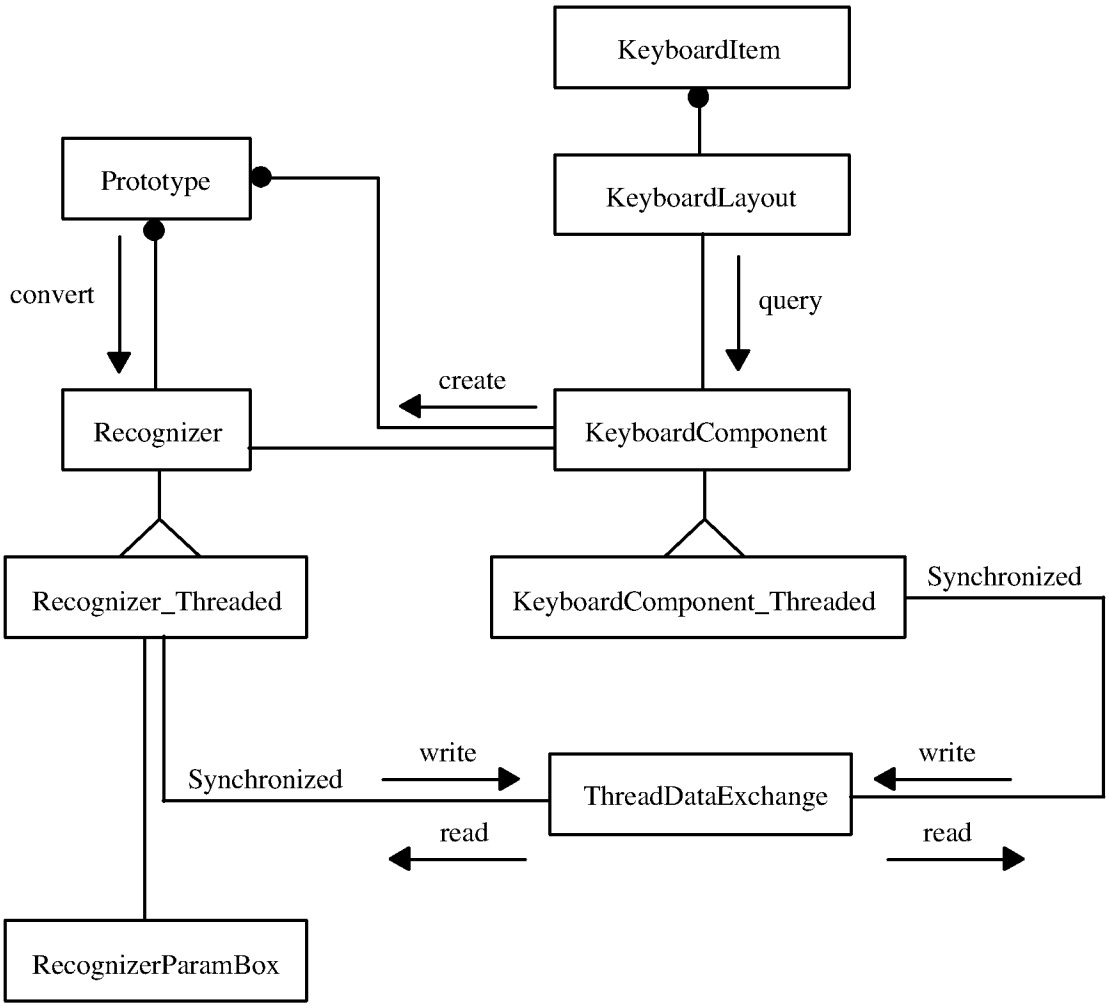


Figure B-3 UML chart of the SHARK system.

Appendix C
SHARK API

The SHARK API consists of 8 classes. It is entirely written in Sun Java™, with the 1.4.0 Sun Java™ API in mind, although the SHARK API at current state have been successfully ran under Sun Java™1.3.1 API and later.

We list a summary of the documentation around these classes as a reference for future implementations of similar systems. Because of space, some methods, field data and inherited fields and methods have been left out.

Recognizer

The `Recognizer` contains an internal database of prototypes and methods for preprocessing and retrieving the most similar prototype from an output signal.

Methods

`addPrototype`	Adds a Prototype object.
`getCandidate`	Retrieves best matching prototype from a input signal.
`getCandidateList`	Returns a sorted list.
`getParameters`	Returns a RecognizerParamBox.
`getPrototype`	Returns a Prototype object.
`removeAllPrototypes`	Clears the database.
`removePrototype`	Remove a Prototype object.
`setParameters`	Inserts parameters from a RecognizerParamBox object
`toString`	Returns a String representation of this object.

Recognizer_Threaded

The `Recognizer_Threaded` is threaded wrapper class for Recognizer. It contains additional means of communicating with the threaded counterpart of `KeyboardComponent`.

Methods

`run`	The run() method of the thread.

KeyboardComponent

`KeyboardComponent` is the core GUI component of SHARK. It is responsible for drawing the keyboard and pen traces, capturing user events and communicating with the `Recognizer`.

The SHARK system is event driven. Therefore, the implementation attaches an `ActionListener` to the `KeyboardComponent` object. When user text entry data is processed the `KeyboardComponent` object will notify all objects listening with an `ActionEvent` object containing the text entry data.

Methods

`addActionListener`	Attaches an ActionListener to this object.
`getLayout`	Retrives the current KeyboardLayout.
`setLayout`	Sets the KeyboardLayout.
`toString`	Returns a String representation of this object.

KeyboardComponent_Threaded

This is a threaded wrapper class for `KeyboardComponent`.

Methods

`run`	The run() method of the thread.

KeyboardLayout

`KeyboardLayout` contains the keyboard layout for the soft keyboard. The keyboard layout is defined as a set of non-overlapping `KeyboardItem` objects. The row and column size of a `KeyboardLayout` object may be set to an arbitrary value.

Methods

`addItem`	Adds a KeyboardItem.
`getColumns`	Returns the columns.
`getItem`	Returns a KeyboardItem.
`getRows`	Returns the rows.
`setColumns`	Sets the columns.
`setRows`	Sets the rows.
`size`	Returns the total number of KeyboardItem objects in this object.
`toString`	Returns a String representation of this object.

KeyboardItem

`KeyboardItem` is an object that describes the properties of a single key. A key may be defined at an arbitrary positive height or width, and positioned anywhere on the canvas dedicated to the soft keyboard. The keyboard layout is automatically scaled, optionally retaining aspect ratio of the individual keys. At present the font size is fixed to the font size specified on the specific KeyboardItem.

Methods

`getColor`	Returns the color.
`getFont`	Returns the font.
`getIdent`	Returns the unique identification String for this object.
`getLabel`	Returns the label.
`getPosition`	Returns the position.
`getSize`	Returns the size.
`setColor`	Sets the color.
`SetFont`	Sets the font.
`setIdent`	Sets the unique identification Strng for this object.
`setLabel`	Sets the label.
`setPosition`	Sets the position.
`setSize`	Sets the size.

RecognizerParamBox

`RecognizerParamBox` is a data structure containing configuration settings for the Recognizer.

Fields

Constants	
`DI_DISTANCE`	Destructive interpolation method.
`NDI_DISTANCE`	Non-destructive interpolation method.
`NPP_DISTANCE`	Normalized point-to-point distance method.
`PP_DISTANCE`	Point-to-point distance method.
Variables	
`max_angle`	Maximum allowed angle.
`min_distance`	Minium allowed distance.
`recognition_method`	Recognition method.

Prototype

`Prototype` is a data structure that holds the prototype data for a single prototype shape or gesture.

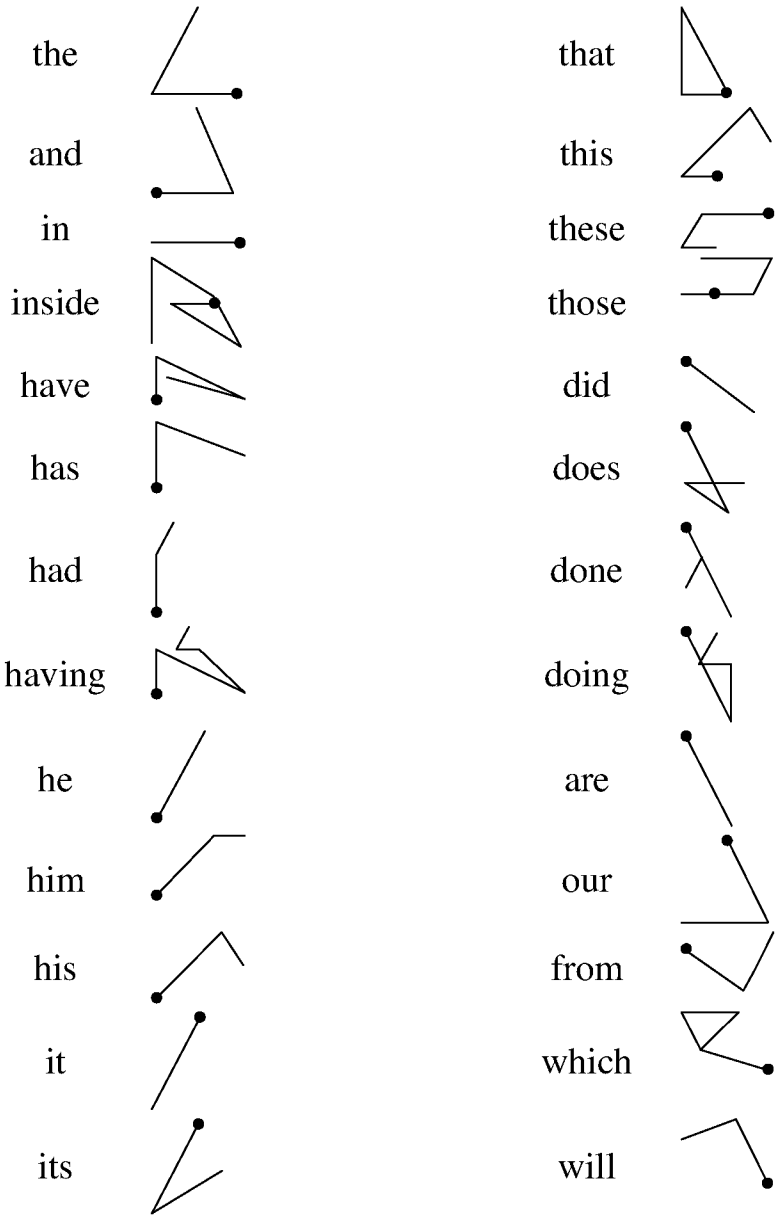
Fields

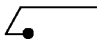
`ident`	A unique identification String for this Prototype object.
`ptData`	A double array containing ordered point data for a prototype.

Appendix D

Words Used in the User Study

Words within top 50 rank in BNC



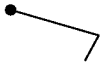
they 

them 

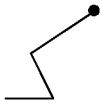
was 

their 

not 

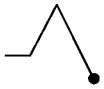
for 

you 

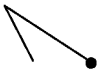
your 

she 

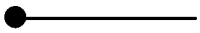
her 

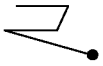
with 

on 

were 

said 

can 

whose 

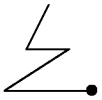
went 

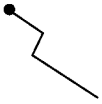
gone 

other 

another 

being 

seeing 

knew 

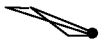
Words within top 300 rank in BNC

knowing 

about 

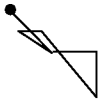
could 

think 

people 

after 

right 

because 

between 

before 

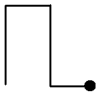
through 

place 

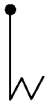
become 

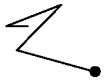
such 

change 

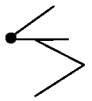
point 

system 

group 

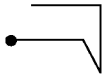
while 

problem 

against 

service 

never 

house 

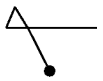
down 

school 

report 

start 

country 

really 

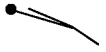
provide 

local 

member 

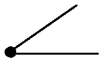
within 

always 

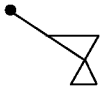
follow 

number 

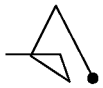
however 

again 

world 

course 

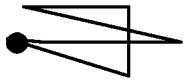
company 

without 

during 

bring 

although 

example 

question 

Appendix E

Top 100 Words in the BNC

Word	SHARK	Frequency	% of BNC
the		61847	6,1847
of		29391	2,9391
and		26817	2,6817
a		21626	2,1626
in		18214	1,8214
to		16284	1,6284
it		10875	1,0875
is		9982	0,9982
to		9343	0,9343
was		9236	0,9236
I		8875	0,8875
for		8412	0,8412
that		7308	0,7308
you		6954	0,6954
he		6810	0,681
be*		6644	0,6644
with		6575	0,6575
on		6475	0,6475
by		5096	0,5096
at		4790	0,479
have*		4735	0,4735
are		4707	0,4707
not		4626	0,4626
this		4623	0,4623
's		4599	0,4599
but		4577	0,4577
had		4452	0,4452

they		4332	0,4332
his		4285	0,4285
from		4134	0,4134
she		3801	0,3801
that		3792	0,3792
which		3719	0,3719
or		3707	0,3707
we		3578	0,3578
's		3490	0,349
an		3430	0,343
~n't		3328	0,3328
were		3227	0,3227
as		3006	0,3006
do		2802	0,2802
been		2686	0,2686
their		2608	0,2608
has		2593	0,2593
would		2551	0,2551
there		2532	0,2532
what		2493	0,2493
will		2470	0,247
all		2436	0,2436
if		2369	0,2369
can		2354	0,2354
her*		2183	0,2183
said		2087	0,2087
who		2055	0,2055
one		1962	0,1962
so		1893	0,1893
up		1795	0,1795
as		1774	0,1774
them		1733	0,1733
some		1712	0,1712
when		1712	0,1712
could		1683	0,1683
him		1649	0,1649
into		1634	0,1634
its		1632	0,1632
then		1595	0,1595
two		1561	0,1561

out		1542	0,1542
time		1542	0,1542
my		1525	0,1525
about		1524	0,1524
did		1434	0,1434
your		1383	0,1383
now		1382	0,1382
me		1364	0,1364
no		1343	0,1343
other		1336	0,1336
only		1298	0,1298
just		1277	0,1277
more		1275	0,1275
these		1254	0,1254
also		1248	0,1248
people		1241	0,1241
know		1233	0,1233
any		1220	0,122
first		1193	0,1193
see		1186	0,1186
very		1165	0,1165
new		1145	0,1145
may		1135	0,1135
well		1119	0,1119
should		1112	0,1112
her*		1085	0,1085
like		1064	0,1064
than		1033	0,1033
how		1016	0,1016
get		995	0,0995
way		958	0,0958
one		953	0,0953
our		950	0,095

Appendix F

Speed Distribution in the User Study

Sorted alphabetically

Word:	Average speed (ms):	Word:	Average speed (ms):
about	1422,281	knowing	1330,64
after	765,6111	member	651
again	959,3	never	2298,5
against	1260,333	not	1754,77
also	1086	number	2034
and	1403,727	on	1867,78
another	1624,486	other	1733,25
are	2192,741	our	1958,23
because	1626,077	people	1401,75
become	2503,1	place	1425
before	2275,077	point	1293
being	3731,676	problem	1495,71
between	2360,313	really	422
can	2301,194	report	1463,33
change	2137,3	right	939,313
company	2738,857	said	3192,97
could	2726	school	3492,25
country	1015	seeing	5420,43
course	1070,3	service	19090,8
did	914,3103	she	7097,04
does	1132,833	start	9359
doing	1137,033	such	6121
done	1467,133	system	8771,09
down	2078,4	that	6720,52
for	1459,333	the	7667,94

Word:	Average speed (ms):	Word:	Average speed (ms):
from	1635,161	their	10278,1
gone	1961,121	them	5162,67
good	2429,5	these	1655,66
group	1511,273	they	2094,49
had	2075,675	think	2394,85
has	1604,606	this	2219,46
have	1872,061	those	2547,63
having	1494,471	through	2372
he	697,5128	was	2194,12
her	676,28	went	1615,97
him	570,1765	were	1301,03
his	758,3103	which	1335,67
house	391,8333	while	1661,33
however	829,4545	whose	1693,78
in	744,375	will	1766,1
inside	971,5882	with	1930,27
it	936,6563	world	2310,67
its	1127,909	you	1857,52
knew	1143,308	your	1869,88

Sorted by top-speed

Word:	Average speed (ms):	Word:	Average speed (ms):
house	391,8333	whose	1693,78
really	422	other	1733,25
him	570,1765	not	1754,77
member	651	will	1766,1
her	676,28	you	1857,52
he	697,5128	on	1867,78
in	744,375	your	1869,88
his	758,3103	have	1872,06
after	765,6111	with	1930,27
however	829,4545	our	1958,23
did	914,3103	gone	1961,12
it	936,6563	number	2034
right	939,3125	had	2075,68
again	959,3	down	2078,4
inside	971,5882	they	2094,49
country	1015	change	2137,3
course	1070,3	are	2192,74
also	1086	was	2194,12
its	1127,909	this	2219,46
does	1132,833	before	2275,08
doing	1137,033	never	2298,5
knew	1143,308	can	2301,19
against	1260,333	world	2310,67
point	1293	between	2360,31
were	1301,029	through	2372
knowing	1330,636	think	2394,85
which	1335,667	good	2429,5
people	1401,75	become	2503,1
and	1403,727	those	2547,63
about	1422,281	could	2726
place	1425	company	2738,86
for	1459,333	said	3192,97
report	1463,333	school	3492,25
done	1467,133	being	3731,68
having	1494,471	them	5162,67
problem	1495,714	seeing	5420,43

Word:	Average speed (ms):	Word:	Average speed (ms):
group	1511,273	such	6121
has	1604,606	that	6720,52
went	1615,966	she	7097,04
another	1624,486	the	7667,94
because	1626,077	system	8771,09
from	1635,161	start	9359
these	1655,661	their	10278,1
while	1661,333	service	19090,8

Appendix G
Questionnaire Used in the User Study

Name: Male Female

Age group: 15-20, 21-25, 26-30, 31-35, 36-40, 41-45, 45-50, 50+

Comments on: the writing method, learning method, fun/boring, fatigue, like-dislike, whatever comes to your mind.

Day 1

Day 2

Day3

Day 4

Final questions (Day 5)

Please rate the writing method:

(Frustrating)	-3	-2	-1	0	1	2	3	(satisfying)
(Dull)	-3	-2	-1	0	1	2	3	(stimulating)
(Difficult)	-3	-2	-1	0	1	2	3	(easy) to use

(Difficult) -3 -2 -1 0 1 2 3 (easy) to learn

If such a method is made available for practical use, would you learn it?

(Definitely no) -3 -2 -1 0 1 2 3 (Definitely yes)

If such a method is made available for practical use, would you use it?

- Not at all
- Yes when there is no physical keyboard
- Yes, to replace keyboard typing sometimes.
- Yes, to replace keyboard typing all the time.

Open comments:

Avdelning, Institution
Division, Department

Institutionen för Datavetenskap
581 83 LINKÖPING

Datum
Date
2002-08-28

Språk
Language
Svenska/Swedish
X Engelska/English

Rapporttyp
Report category
Licentiatavhandling
Examensarbete
C-uppsats
X D-uppsats
Övrig rapport
—

ISBN

ISRN LIU-KOGVET-D--02/07--SE

Serietitel och serienummer **ISSN**
Title of series, numbering

URL för elektronisk version
http://www.ep.liu.se/exjobb/ida/2002/007/

Titel
Title
Design och utvärdering av ett mjukvarutangentbord med stenografiskt stöd.

Design and Evaluation of a Shorthand Aided Soft Keyboard.

Författare
Author
Per-Ola Kristensson

Sammanfattning
Abstract
A major drawback of mobile computing is the lack of an efficient text entry method. Previous attempts of handwriting/ gesture recognition or stylus soft keyboards have been shown to be inefficient, inaccurate or very hard to learn. In this work, we utilize the ATOMIK alphabetically tuned and optimized stylus soft keyboard and introduce the concept of word-level shorthand strokes formed after the corresponding keys on the keyboard layout. Since the gestures are scale and partially location independent, they require little or no visual attention from the user. An implication of Zipf s law is that the most common words make up a large percent of the text mass - in the 100 million words large British National Corpus the 100 most common words make up 45% of the entire corpus. Hence the largest performance gain is acquired by introducing gestures for only a limited subset of the English language and utilizing the soft keyboard for the rest. A gesture recognition engine based on elastic matching was developed and a working prototype system was used in a longitudinal pilot study involving six subjects. The study showed that the users learned about 58 shorthand gestures on average after four sessions of 40 minutes practice. Their rate of learning was rather constant across sessions, acquiring about 15 new words in each session of practice.

Nyckelord
Keyword
shorthand, keyboard, soft keyboard, stylus keyboard, elastic matching, mobile text entry, text input, text entry, stenography, gesture, pda, small devices, palm top